2006

Visualization of Secondary RNA Structure Prediction Algorithms

Brandon Hunter
San Jose State University

Visualization of Secondary RNA Structure Prediction Algorithms

A Project Report

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Brandon Hunter

2006

Contents

Chapter 1

Introduction

This chapter introduces the secondary structure prediction problem. It describes what the secondary structure prediction problem is and why it is important. Based on the importance of the algorithm it is essential to have a clear means to visually represent the problem. Therefore, this chapter details the high level goals of the visualization. It details how the visualization will visually represent the problem through several simultaneous representations. These visual representations will be tied together in order to increase the understanding of the algorithm.

1.1 RNA Secondary Structure Prediction

Secondary RNA structure prediction is a problem that endeavors to predict the two dimensional structure of an RNA sequence given only the nucleotide bases of the sequence. This problem is very complex because there are an enormous number of possible secondary structures that could be created from any given RNA sequence. As the length of the RNA sequence gets longer, so too does the number of possible secondary structures that could be created from the sequence. The key to the problem of secondary RNA structure prediction is to distinguish between structures which are biologically correct from those that are not correct. In practice, the algorithms that attempt to solve the secondary RNA structure prediction problem are still being perfected but they all suffer computational complexity issues.

1.2 Importance of RNA Secondary Structure

The RNA structure prediction problem is very important to the Bioinformatics field because one very common task performed by researches is to find all the homologues of a given RNA sequence. The way this is typically done is that a researcher would search a genome database for matches to a sequence and if a similarity threshold were achieved then the sequences would be considered homologous. The database search could be significantly improved if one were able to search for similarities in secondary structure in addition to the sequence itself. This improvement can be achieved because it has been shown that homologous RNA can share secondary structure while not sharing a high degree of sequence similarity.

1.3 RNA Secondary Structure Visualization

The RNA secondary structure problem has been such an important topic that many techniques have been devised to visualize the secondary structure. Instead of creating a visualization where one simply enters the input RNA sequence and the program calculates a visual representation of the output, this paper proposes a visualization which is richer in features and visual representations. Instead of just being an input/output visualization this paper details a visualization where the individual can step through an RNA sequence and interactively view the secondary structure as it's created. The visualization detailed in this paper is intended to be a teaching tool which would help individuals who are new to the topic quickly and intuitively understand how the secondary structure of a particular RNA is calculated. Since the focus of this visualization is to help increase understanding it follows that the definition of visualization used in this paper is as proposed by Card, Mackinlay and Shneiderman in "Readings in Information Visualization, Using Vision to Think" as: "The use of computer-supported, interactive, visual representations of abstract data to amplify cognition" [4]. The authors of the book propose that the purpose of the visualization should be to amplify cognition or the understanding of a problem. That is exactly the intention of the visualization in this paper. The visualization will amplify cognition by

allowing the individual to visually see the connection between the algorithms matrix used in the secondary structure calculation to the visual representation of the RNA. This visualization will use several techniques to increase cognition all of which were proposed in the book [4]. First, the user of the visualization will have increased memory and processing resources available because they will not have to use much thought to understand the visualization therefore freeing the user to spend resources elsewhere. Second, the user will not have to waste mental resources searching for information because the visualization will provide all relevant information in the display. Third, the visualization will be created in such a way that it enhances the user's ability to find patterns in the information. Fourth, the visualization will allow the user to perceptually infer details about the problem. Fifth, the visualization will use techniques that draw the attention of the user to the desired location. Lastly, the visualization will increase cognition by encoding the details of the problem in a medium which can be manipulated through a forward and back stepping mechanism.

Since the goal of this paper is to detail a means to display several representations of the secondary RNA structure prediction problem, it is important to understand the make up of RNA. The next chapter will detail the make up of RNA as well as all the known types of RNA. By analyzing all the know types of RNA, it will help give an understanding of which types of RNA are important to study. It is also important to know the typical sequence lengths for each type of RNA in order to know which types of RNA this visualization will be able to model. Since this visualization is intended as a teaching tool it will not in devour to visualize extremely long sequences. The intention is to focus on conveying as much information as possible about how the secondary structure algorithm works. Therefore, the target sequence length is approximately 100 nucleotides or less.

Chapter 2

RNA Introduction

This chapter gives an overview of the components of RNA and starts by describing the actual make up of RNA at the molecular level. It describes the three main structural components of RNA, the nitrogenous base, the pentose sugar and the phosphate group. These three structural components form chains in RNA so it is shown how the chains are bound together. Finally, the chapter describes the types of RNA. Each type of RNA is categorized based on similar function and sequence length.

2.1 What is RNA

RNA is a biological structure contained within all living organisms. The term RNA is actually an acronym which stands for ribonucleic acid. There are two types of nucleic acids found within living organisms. There's deoxyribonucleic acid and ribonucleic acid. The nucleic acid is composed of chains of nucleotides. The nucleotides are further composed of three components: a nitrogenous base, a pentose sugar and a phosphate group **(See Figure 2.1) [25]**.

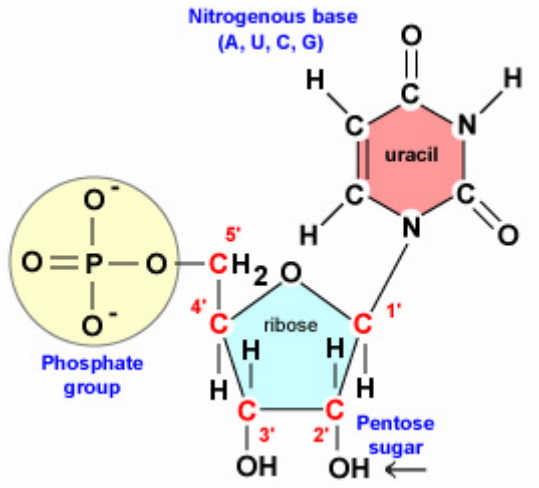


Figure 2.1. RNA Nucleotide [25]

Notice that in figure 2.1 above, if the oxygen molecule where the arrow points was removed then this sugar would become a deoxyribose sugar. This is where the deoxyribose name is derived from since it is a ribose sugar without the oxygen molecule at the indicated location.

2.1.1 Nitrogenous Bases

There are five primary nucleotide bases which are common to both DNA and RNA, Adenine, Guanine, Cytosine, Thymine and Uracil [7]. Adenine, Guanine, Cytosine and Thymine are most commonly found in DNA while Thymine is replaced with Uracil in RNA. These nucleotide bases are generally grouped according to the number of atoms that make up the ring in the nucleotides structure. Purine bases have a fused ring composed of 9 atoms, 5 carbon atoms and 4 nitrogen atoms [7]. Adenine and Guanine are Purine bases. Pyrimidine bases have a single ring composed of 6 atoms, 4 carbon and 2 nitrogen [7]. Cytosine, Thymine and Uracil are all Pyrimidine bases **(See Figure 2.2)**.

All nucleotide bases form bonds to other nucleotide bases in specific combinations. These bonds are generally referred to as Watson-Crick base pairs [7]. The Watson-Crick base pairing states that Adenine forms a bond with its complementary base pair Thymine and that Cytosine forms a bond with its complementary base pair Guanine in DNA. In RNA, the Thymine is replaced with Uracil so the complementary pair is Adenine to

Uracil. Adenine bonds to either Thymine or Uracil with two hydrogen bonds while Cytosine bonds to Guanine with three hydrogen bonds. Since the Cytosine and Guanine pair has more bonds it is considered to be a more stable paring.

The five primary bases are only a small fraction of the total number of bases which actually appear in RNA. There are an abundance of modified bases such as Pseudouridine and Thymidine which generally occur in the TΨC loop of tRNA [7]. There are around 100 modified bases which occur in RNA, and there are so many modified bases that most of them are not generally understood [7].

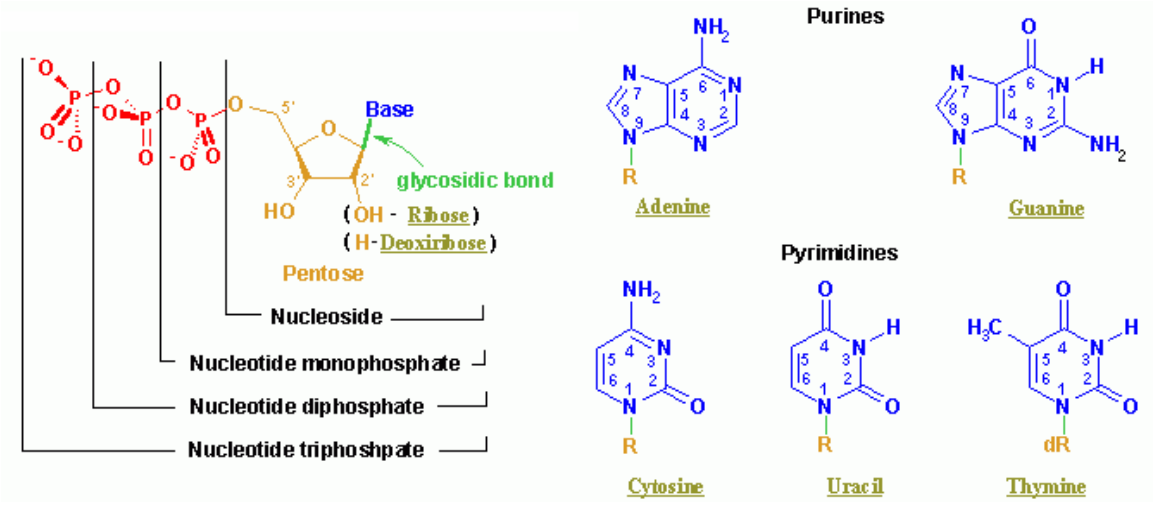


Figure 2.2. RNA Bases [7]

2.1.2 Pentose Sugar

The pentose sugar is a biological structure with five carbon atoms. The root of the name pentose, pent, meaning five suggests as much. There are many types of pentose sugars such as Lyxose, Xylose, Arabinose, and Ribose but RNA contains a ribose sugar [7]. The ribose sugar in RNA has five carbon atoms which are typically labeled **1'** through **5'** with hydroxyl (OH) groups at the **1'**, **2', 3'** and **5'** locations **(See Figure 2.1)**. DNA differs from RNA in that DNA is missing a hydroxyl group at the **2'** location. The ribose sugar of an RNA chain is located in between two phosphates while the nitrogenous base forms bonds on the side. The phosphate bonds to the sugar at the **5'** and **3'** locations while the nitrogenous base bonds at the **1'** location **(See Figure 2.3)**. The phosphate

bonds at the **5'** and **3'** locations are what allow the RNA to forms chains. The chain starts with a phosphate on the **5'** side of the sugar followed by the sugar with a nitrogenous base at the **1'** side of the sugar and then another phosphate at the **3'** side of the sugar. The phosphate on the **3'** side of the sugar will bond with the **5'** side of the next sugar and the thus the chain is formed [26].

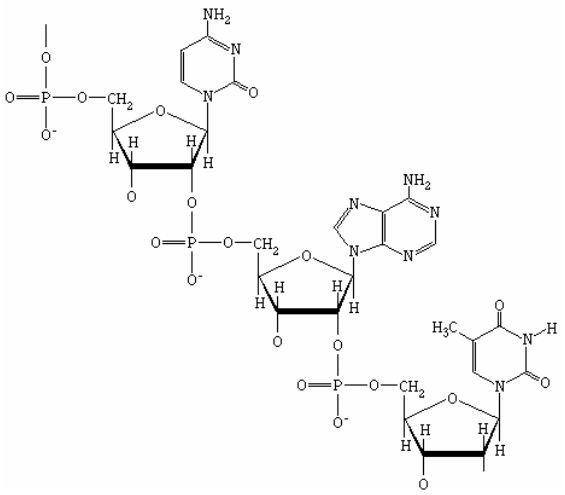


Figure 2.3. RNA Chain [26]

2.1.3 Phosphate Group

The phosphate consists of a central phosphorus atom surrounded by four oxygen atoms [7]. The phosphate acts as the linking mechanism between the ribose sugars in an RNA chain **(See Figure 2.3)**.

2.2 RNA Synthesis

Although there are many types of RNA, each of which perform a specific function, one of the main purposes of RNA is to be the messenger which helps carry the genetic information coded in the DNA and to convert that coded information into proteins. The Central Dogma of Molecular Biology states just that [2]. The Central Dogma states that DNA can replicate itself; it can transition to RNA through a process called translation and it can transition to a protein through a process called transcription.

The synthesis of the RNA which occurs in the transcription phase of the central dogma is performed in the following manner. First, an RNA polymerase binds to the DNA at a promoter site which is typically found upstream of the gene that is going to be transcribed [7]. Next, the DNA is unwound by the polymerase into two individual sequences referred to as the coding strand and the template strand. The coding strand goes in the **5'** to **3'** direction while the template goes in the **3'** to **5'** direction. The RNA is synthesized from the template strand so the polymerase travels down the template strand in the **3'** to **5'** direction. The RNA is created by continually adding nucleotides to the RNA sequence which are complementary to the template strand in a process referred to as elongation [7]. This process will continue until a code in the DNA tells the polymerase to stop.

2.3 Types of RNA

Biologists have discovered that there are several types of RNA which fall into categories based on their biological function and typical sequence length. For the purposes of this visualization it is very important to understand the multitude of RNA types in order to understand which sorts of RNA the visualization will be able to handle. It is also important to understand the function of each type of RNA so that it can be determined if any type of RNA has more significance to the secondary structure prediction problem than another.

2.3.1 Messenger RNA (mRNA)

Messenger RNA (mRNA) is responsible for carrying the genetic information encoded in the DNA to ribosome receptor sites where the genetic information is translated into a protein [7]. The mRNA classes of RNA's are quite long and are typically in the order of several thousand base pairs. In eukaryotic cells the mRNA does not start the process of protein synthesis until the mRNA has exited the nucleus and is in the cytoplasm. In prokaryotic cells there is no nucleus so protein synthesis can begin as the mRNA is being transcribed from the DNA. Once in the cytoplasm, the mRNA attaches to an organelle called a ribosome. The ribosome helps the transfer RNA (tRNA) attach to the mRNA three base pairs at a time referred to as a codon [2]. The mRNA is read in order while the

tRNA connected to the mRNA one codon at a time. Each tRNA has an attached amino acid that forms a chain as the mRNA is read and the amino acid chain becomes a protein.

2.3.2 Transfer RNA (tRNA)

Transfer RNA (tRNA) is a short sequence of about 74 to 93 nucleotides in length [7]. The tRNA generally has a secondary structure that resembles a clover leaf where the middle clover leaf branch contains three nucleotides referred to as the anti-codon. The **3'** end of the tRNA usually ends with the three nucleotide sequence CCA and then is attached to an amino acid. The tRNA is responsible for binding to the mRNA at the tRNA anti-codon site through hydrogen boding [7]. As the tRNA bonds to the mRNA at the ribosome bond site, the tRNA connectes its amino acid to the growing polypeptide chain and hence facilitates the synthesis of proteins.

2.3.3 Ribosomal RNA (rRNA)

Ribosomal RNA (rRNA) is a component of the protein synthesis molecule called the ribosome. There are 4 types of rRNA in eukaryotic cells referred to as 18S, 5S, 5.8S and 28S [7]. The 18S rRNA along with around 30 different proteins constitute the small subunit of the ribosome while the 5S, 5.8S and 28S rRNA's along with around 45 different proteins constitute the large subunit of the ribosome. The "S" after the name of each of the rRNA types refers to Svedberg units which are the units used to measure sediment in the ultracentrifuge. The numbers in the rRNA names are not proportional but they reflect the size of the rRNA molecule [11].

2.3.4 Small Nuclear RNA (snRNA)

Small Nuclear RNA (snRNA) is a short RNA sequence generally in the range of 60 to 300 nucleotides long [12]. The snRNA has several functions but is generally connected to the processing of other types of RNAs. One example of how snRNA is connected to the processing of other RNA's is that snRNAs are connected with ploydenylation or with the terminating of the mRNA at the Poly-A tail on the **3'** end of the RNA [12].

Another example is snRNA's are connected to the spliceosome which functions to exclude the interons and connect the exons in the final mRNA [12].

2.3.5 Double Stranded RNA (dsRNA)

Double Stranded RNA (dsRNA) is a form of RNA which has two connected strands. The two strands are complementary, similar to what is found in DNA [7]. Double stranded RNA acts as a mechanism to initiate the process of RNA interference (RNAi). RNA interference occurs when small subsequences in the dsRNA, which are homologous to sequences in a gene, interfere with the expression of the gene. From what is currently known the dsRNA is cut into its individual strands by an enzyme known as a dicer [7]. The individual strands then bond with other RNA sequences which are complementary to the single dsRNA strand which renders the RNA useless. Double stranded RNA is also known to be a component in the formation of small interfering RNA (siRNA) as well as a component of the genetic material in some viruses [7].

2.3.6 Small Interfering RNA (siRNA)

Small Interfering RNA (siRNA) are short RNA sequences in the range of 20 to 25 nucleotides long [7]. Small interfering RNA is a double stranded RNA sequence which is a component of RNA interference (RNAi) (See Double Stranded RNA)

2.3.7 Micro RNA (miRNA)

Micro RNA (miRNA) are short single stranded RNA sequences in the range of 20 to 25 nucleotides long [7]. Micro RNA functions as a mechanism to regulate gene expression. The miRNA is complementary to a portion of an mRNA which it will bond to in order to regulate gene expression. When the miRNA bonds to the mRNA it inhibits that portion of the RNA from being translated into a protein [7].

2.3.8 Other RNA Types

Although the RNA types described so far tend to be the most well known and understood types of RNA, there are many other types of RNA such as Guide RNA (gRNA), Efference RNA (eRNA), Signal Recognition Partical RNA (srpRNA), Phages RNA (pRNA), Transfer Messenger RNA (tmRNA) and many others [12].

2.4 RNA World Hypothesis

The RNA World Hypothesis is a theory that suggests RNA is the precursor that made it possible for DNA to be created. This hypothesis was originally suggested by Carl Woese in his 1967 book titled ***The Genetic Code*** [9] [10]. The RNA world hypothesis arises from a "chicken-and-egg" type problem that was present in the primordial earth. How would it have been possible for DNA to have been created when it takes proteins to replicate and transcribe DNA? On the other hand, how would it have been possible to have the proteins when DNA is the mechanism that carries the genetic code which is needed to synthesize the proteins? Each is required to synthesize the other. RNA seemed to be the perfect answer to this problem. Since RNA can store genetic information and since its single stranded structure allows it to form many tertiary structures similar to proteins it might have been possible that RNA did the job of both DNA and protein. In the early 1980's came the discovery that made this hypothesis seem very plausible. It was discovered that there are some RNA's which are self replicating. This prompted Walter Gilbert to write in 1986, "One can contemplate an RNA World" [9], hence, the theory was coined the "RNA World Hypothesis".

2.5 Implications of World Hypothesis on Structure Prediction

The implications of the RNA World Hypothesis on secondary RNA structure prediction are that all living organisms could have evolved into their present form from a common type of RNA. This could mean that there are many similarities in the RNA of present day organisms. Even if the sequence of the RNA in current organisms has varied a sufficient amount so as that is would seem they are not related, it might be possible that the RNA actually has a similar secondary or tertiary structure which effectively have the same

function. This is why researchers are very interested in finding computer based algorithms that can accurately predict the secondary structure of RNA. If a database can be created that describes the secondary structure of RNA sequences, then the secondary structure could be used in addition to just using the sequence when performing tasks such as trying to determine if several RNA sequences are homologous.

Since the visualization is intended to be a teaching tool and is not focused on dealing with extremely long RNA sequences, one can determine which RNA classes the program will be useful to work with. The visualization is intended to deal with sequences of 100 base pairs or less. By analyzing the typical lengths of each class of RNA one can determine that the visualization will be able to handle tRNA's, some of the shorter snRNA's, siRNA's and miRNA's. Of course the program could always visualize short subsequences of longer RNA's such and mRNA. Now that it has been shown which type of RNA the visualization will be able to handle, the next chapter will give a formal description of the secondary structure prediction problem and will discuss the types of algorithms which have been discovered which attempt to solve the secondary structure prediction problem.

Chapter 3

Secondary RNA Structure

This chapter gives an overview of the secondary RNA structure prediction problem. It starts out by formally describing the secondary structure prediction problem as suggested by Zuker [14]. It describes the different types of algorithms and the classifications that these algorithms fall into. The chapter explains the issues and deficiencies of the algorithms. There is a section which describes the structural elements that the secondary structure is made of. Finally, the chapter describes all ways in which the secondary structure has been visualized.

3.1 Secondary Structure Formal Description

RNA secondary structure refers to the two dimensional shape that RNA would physically fold into under natural conditions. As RNA folds back on itself it forms hydrogen bonds at complimentary base pair locations. These hydrogen bonds formed by the pairing of complementary Watson-Crick bases as well as the weaker wobble pair G-U are described as canonical base pairs [14].

Formally, the secondary structure of RNA can be described as suggested by Zuker [14] as follows: An RNA sequence is represented by **R** as $\mathbf{R = r_1, r_2, r_3, \ldots, r_n}$, where $\mathbf{r_i}$ is called the $\mathbf{i^{th}}$ nucleotide. Each $\mathbf{r_i}$ belongs to the set {A, C, G, U}. A secondary structure, or folding, on **R** is a set **S** of ordered pairs, written as **i,j**, **1 <= i <= j <= n** satisfying:

1. **j – i > 4**
2. If **i,j** and **i',j'** are 2 base pairs, (assuming without loss in generality that **i <= i'**), then either:
    a. **i = i' and j = j'** (they are the same base pair),
    b. **i < j < i' < j'** (**i,j** precedes **i', j'**), or
    c. **i < i' < j' < j** (**i,j** includes **i',j'**).

Item 2c above disallows pseudoknots which occur when two base pairs, **i,j** and **i',j'** satisfy the condition **i < i' < j < j'**[14]. The formal description does not account for pseudoknots for several reasons. First, the algorithms which try to predict the secondary structure through energy minimization are not able to handle pseudoknots. Energy minimization algorithms can not handle pseudoknots because it is beyond current scientific understanding how to assign energy values to the structures created by psuedoknots. Secondly, pseudoknots are not considered because the dynamic programming based algorithms are not able to handle the loop structures created by pseudoknots.

3.2 Taxonomy of Algorithms

There are several methods used in the laboratory in order to determine the secondary and tertiary structure of RNA. These methods include x-ray crystallography and nuclear magnetic resonance spectroscopy [15]. The problem with these methods is that they are very expensive and time consuming to produce the secondary structure results. It would be tremendously preferable if a computer based algorithm could be created which would accurately calculate the secondary structure in mere seconds. This has been the focus of many researches for the past several decades. There are many types of algorithms which have been devised which endeavor to fulfill this goal but the algorithms can be categorized into two main types, deterministic and stochastic [13].

3.2.1 Deterministic

There are a whole host of algorithms which fit into the classification of deterministic. The one fact that is common to all algorithms of this type is that the correct next step in the algorithm only depends on the current state of the algorithm. There is no point in the

algorithm at which there are several next steps that could happen with some unknown way to choose between them. Algorithms that fall into this category are
Minimum Free Energy such as Zuker's algorithm [14], **Kinetic Folding** such as Martinez [16], **5' – 3' Folding** [13], **Partition Function** [13], and **Maximum Matching** such as Nussinov [8]. Kinetic Folding and 5' – 3' Folding are able to determine pseudoknots.

3.2.2 Stochastic

The common theme between all the stochastic algorithms is that they are based on probabilities. One such example is based on a special Monte Carlo procedure know as **Simulated Annealing** [17]. The Simulated Annealing algorithm is able assign probabilities to for both the opening and closing of single base pairs. This allows the algorithm to account for a wide range of secondary structures.

3.3 Issues with Algorithms

The main issue with all the computer based algorithms is that they are no more than first order approximations of the actual secondary structure which would occur in nature. The determination of secondary structure is by no means an exact science. Furthermore, the structure which the algorithm calculates to be the optimal structure might not be the most biologically correct. Many of the algorithms allow for suboptimal structures to be calculated as well taking into account this anomaly.

3.4 Secondary Structure Elements

All RNA secondary structures are composed of several basic structural shapes which occur naturally when RNA folds back on its self. These basic structures are usually depicted as two dimensional pictures which indicate the positions where base pair bonds occur. The regions where base pairs stack on top of each other and form into helical regions are called stems or stacking pairs **(See Figure 3.1 Stacking Pairs)**. Sections of RNA which occur at either the start or end of the sequence that are not part of any structure are called unstructured single strands or free ends **(See Figure 3.1 Joint and**

Free Ends). All other structures formed by RNA are variations of loop structures which occur when a section of RNA loops around on its self and is bounded by base pairs. Hairpin loops are loops which occur at the end of a stem and consist of three or more base pairs because a three base pair loop is the smallest biologically feasible loop **(See Figure 3.1 Hairpin Loop)**. Hairpin loops are sometime referred to as stem loops but the hairpin name stuck because the loops resemble a hairpin. A bulge which occurs in a single strand of a stem is referred to as a bulge loop **(See Figure 3.1 Bulge)**. When bulges occur on both strands of a stem an interior loop is formed **(See Figure 3.1 Interior Loop)**. When loops occur which have three of more branches (stems) extending out of the loop then a multi-branched loop of formed **(See Figure 3.1 Multiple Loop)**. The last type of loop is referred to as a pseudoknot. A pseudoknot occurs when bases inside a loop are bonded with bases in another section of the RNA which is outside the bounding stem of the loop. Psuedoknots occur relatively infrequently as compared to the other RNA structure elements [1][13].

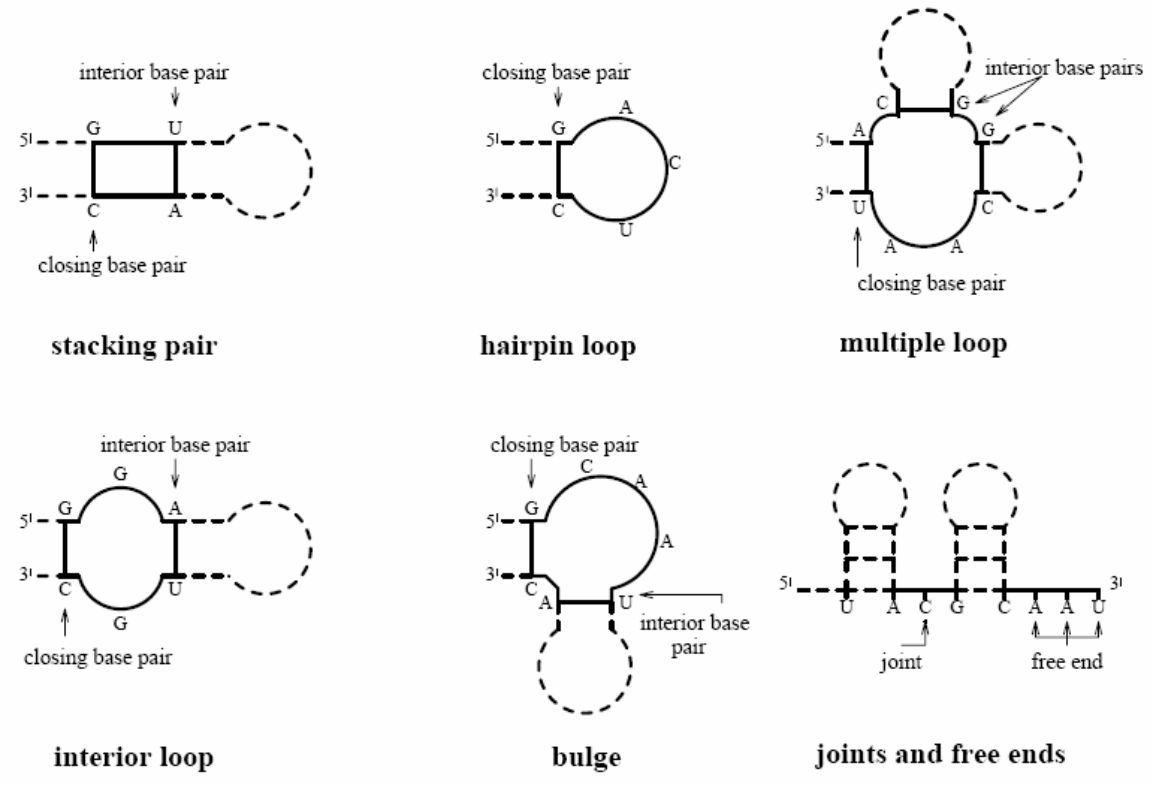


Figure 3.1. Secondary Structure Elements [13]

3.5 Secondary Structure Visualizations

The human mind is not capable of comprehending problems which have large amounts of data in strictly numerical or character formats. The RNA secondary structure problem is one such problem where any given RNA sequence could be hundreds of bases in length and that fact makes it necessary to devise methods to help the human mind comprehend the data. In order to increase the comprehension of the RNA secondary structure problem, many types of visualizations have been devised which represent the data in alternate formats so that the largest amount of intuitive understanding can be gained from the depictions.

3.5.1 String Representation

The simplest form of representing an RNA sequence is strictly by the bases which make up the sequence **(See Figure 3.2)**. A string is formed where the characters in the string represent the four RNA bases. The characters in the string are positioned as to represent the ordering of the bases in the RNA sequence.

AACGGAACCAACAUGGAUUCAUGCUUCGGCCCUGGUCGCG

Figure 3.2. RNA in String Representation

3.5.2 Bracketed Representation

The bracketed representation is one of the simplest ways to visualize the secondary structure of RNA. This representation is sometimes referred to as bracket dot notation because of the brackets and dots used in the representation [31]. The bracketed representation consists of using the string representation of RNA on one line and then on a line directly below the string representation a sequence of open or close brackets and dots are used to represent nucleotides which are bonded as pairs **(See Figure 3.3)**. If a bond exists between nucleotides at position i and position j then an open bracket '(' is used at position i and a close bracket ')' is used at position j to represent the bond. If no bond exists then a dot is placed at the nucleotide position to represent that no bond exists.

For every open bracket there must be a corresponding closing bracket to represent the pairs.

```
AACGGAACCAACAUGGAUUCAUGCUUCGGCCCUGGUCGCG
.(.((..))((((())(().((()))))(((())(.)))(()))
```

Figure 3.3. Secondary Structure in Bracketed Representation

3.5.3 Linked Graph Representation

For the linked graph representation the nucleotide bases are drawn on a line at equidistant intervals. Arcs are then drawn which connect base pairs which have bonds [18]. This representation makes if very easy to determine if pseudoknots exist by examining the graph for arcs that cross one another. If any arcs cross then a psuedoknot exists.

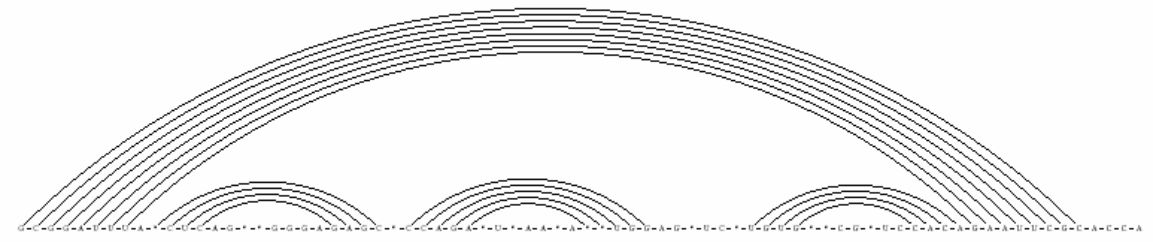


Figure 3.4. Linked Graph Representation of tRNA^{phe} [18]

3.5.4 Planar Graph Representation

The planar graph representation is the most intuitive representation. This is the closest approximation to what the actual two dimensional structures would look under natural conditions. The planar graph is merely a topology therefore structures drawn in the graph which visually seem close together may actually be distant in reality.

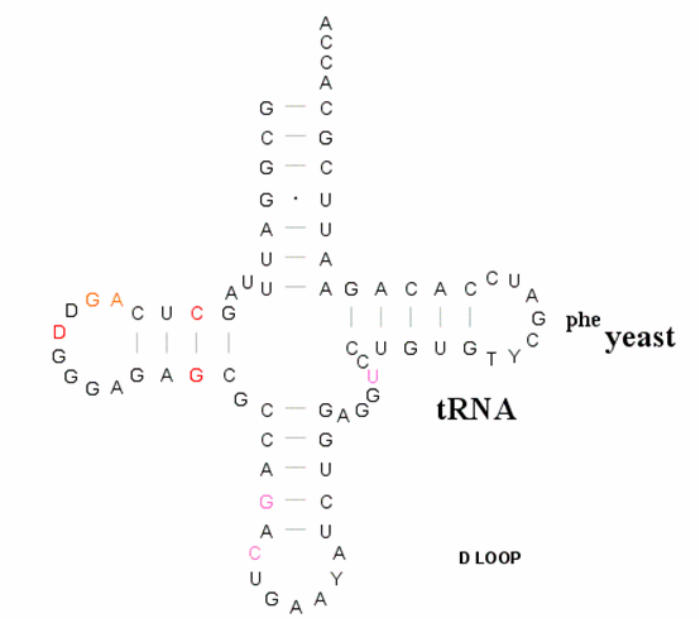


Figure 3.5. Planar Graph Representation of $tRNA^{phe}$ [19]

3.5.5 Tree Representation

The tree representation of RNA secondary structure not only produce a visual display of the secondary structure but also allows mathematical properties of tree theory to be used in the process of examining the tree. The tree graph is actually a forest where paired bases correspond to internal nodes. The labels on the internal nodes are the bases which are paired. The leaf nodes correspond to the unpaired bases whose label is a single base [20]. One useful operation that can be performed on two forest graphs is creating a mathematical value which represents how similar two forests are.

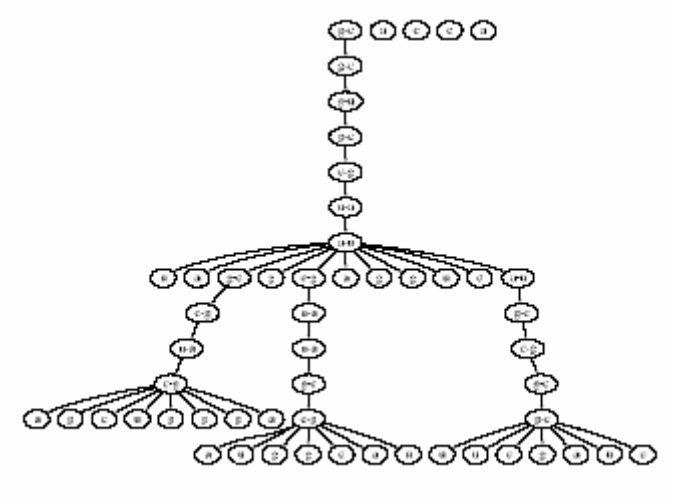


Figure 3.6. Tree Representation of tRNA^{phe} [20]

3.5.6 Circular Representation

The circular representation of secondary RNA structure can be thought of as an extension to the Linked Graph representation where the ends of the string have been wrapped around into a circle [8]. The circular representation uses a circle and then places the nucleotide bases at equidistant intervals around the circle. Chords are then drawn on the interior of the circle between base pairs that form a bond. This representation also allows for easy visual detection of pseudoknots by examining the graph for any chords that intersect. If any chords intersect then a pseudoknot is present. This representation was first devised by Ruth Nussinov [8].

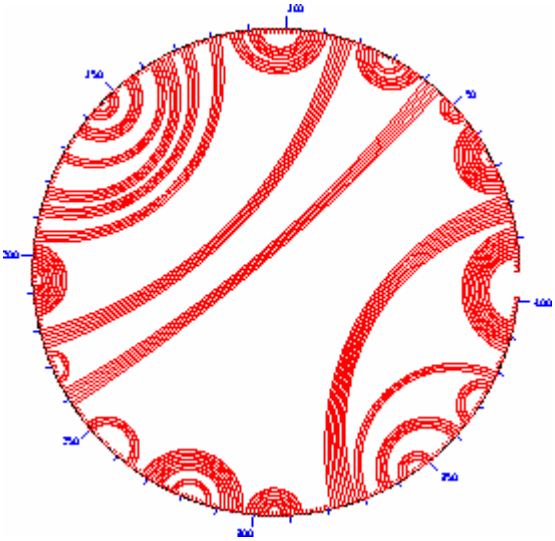


Figure 3.7. Circular Graph Representation [14]

3.5.7 Matrix Representation

The matrix representation is a visual representation of the dynamic programming matrix which is created in algorithms like Nussinov's [1]. The nucleotide bases are listed horizontally and vertically along the edges of the matrix and then the scores from the algorithm fill the interior of the matrix. Some representations show the trace back path through the matrix as a color scale path.

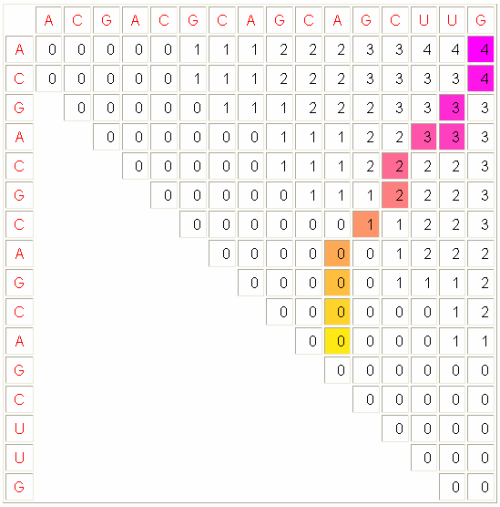


Figure 3.8. Matrix Representation (Dynamic Programming Matrix) [21]

3.5.8 Dot Plot Representation

A dot plot is a graph which is setup as a triangular array. The RNA sequence it places along the to axis of the triangular array and a dot is placed in the graph corresponding to pair i,j at the ith row and jth column. Dot plots are generally used for comparative analysis because many dot plots can be superimposed on the same graph where the plot can be easily compared [14].

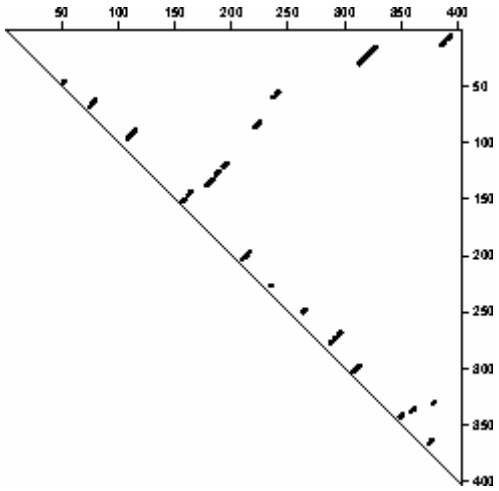


Figure 3.9. Dot Plot Representation [14]

3.5.9 Mountain Plot Representation

The roots of the mountain plot were originally devised in a paper written by Paulien Hogeweg in 1984 [22]. Later, another related paper written by Danielle Konings in 1989 furthered the mountain plot "by identifying '(', ')', and '.', with "up", "down", and "horizontal", respectively" [18]. The mountain plot is represented by three elements: peaks, plateaus, and valleys. The peaks correspond to hairpin loops, the plateaus correspond to unpaired bases and the valleys indicate either unpaired sequences between the branches of a multi-branch loop or unpaired sequences which join components of the secondary structure [18].

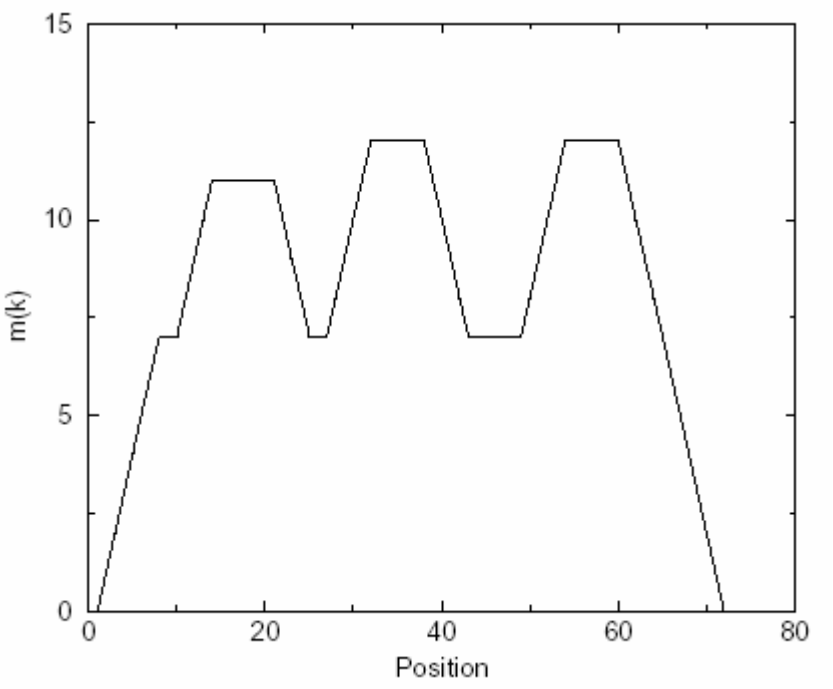


Figure 3.10. Mountain Plot Representation [18]

Now that the RNA secondary structure prediction problem has been formally described, the next chapter will focus specifically on the Nussinov base pair maximization algorithm. There are actually two versions of the Nussinov algorithm, the standard version and the stochastic context free grammar version. The next chapter will detail both versions.

Chapter 4

Nussinov Folding Algorithm

This chapter gives a detailed description of the two versions of the Nussinov algorithm which will be implemented in the visualization. First, the standard Nussinov algorithm is detailed, a formal algorithm definition is described and then the fill and traceback stages are explained. A list of enhancements to the original algorithm are suggested which will be incorporated into the visualization. Second, the stochastic context free grammar version of the Nussinov algorithm is detailed. A formal algorithm definition is described and the fill and traceback stages are explained just as in the standard version.

4.1 Nussinov Algorithm

The Nussinov algorithm is a base pair maximization algorithm. In other words, the algorithm tries to calculate the secondary structure which has the maximum possible number of base pairs. The algorithm was first conceived by Ruth Nussinov in a paper that she wrote in 1978 [8]. The algorithm makes use of a dynamic programming algorithm to recursively find the optimal structure for small subsequences and then recursively works its way out to larger and larger subsequences. At any point during the recursive calculation there are only four possible ways to obtain the optimal structure from the optimal structures of the smaller subsequences **(See Figure 4.1)** [1]. The four ways to obtain the optimal structure as described in the Durbin book are [1]:

1. add unpaired position **i** onto best structure for subsequence **i + 1, j**;
2. add unpaired position **j** onto best structure for subsequence **i, j – 1**;
3. add **i, j** pair onto best structure found for subsequence **i + 1, j – 1**;
4. combine two optimal substructures **i, k** and **k + 1, j**.

When two substructures are combined as in item 4 it is referred to as a bifurcation.

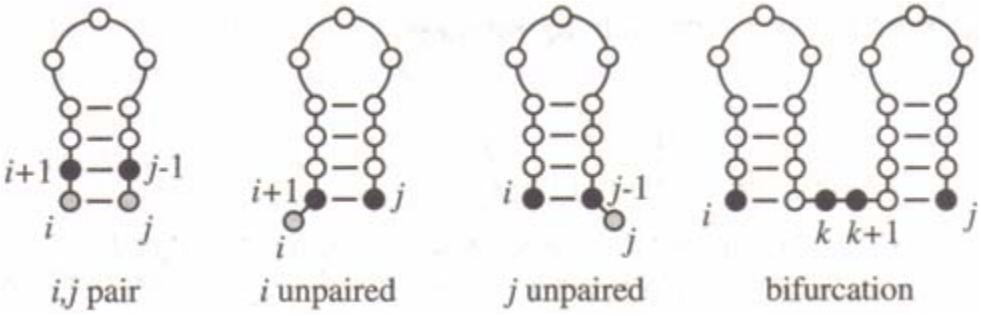


Figure 4.1. Nussinov Structures [1]

4.1.1 Formal Algorithm Definition

Given a sequence $\boldsymbol{x}$ of length **L** with symbols $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_L$. Let $\boldsymbol{\delta(i, j) = 1}$ if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are a complementary base pair; else $\boldsymbol{\delta(i, j) = 0}$. Then recursively calculate scores $\boldsymbol{\delta(i, j)}$ which are the maximal number of base pairs that can be formed for subsequences $\boldsymbol{x}_i,\ldots,\boldsymbol{x}_j$ [1].

4.1.2 Fill Stage

The fill stage of the Nussinov algorithm works by first creating a rectangular array for the dynamic programming matrix. The matrix is initialized so that both the diagonal and the lower diagonal are set to zero. The algorithm then proceeds along the diagonals calculating the score as described below [1].

Initialization:

$$\boldsymbol{\gamma(i, i-1) = 0} \quad \text{for } \mathbf{i = 2} \text{ to } \mathbf{L};$$
$$\boldsymbol{\gamma(i, i) = 0} \quad \text{for } \mathbf{i = 1} \text{ to } \mathbf{L}.$$

Recursion: starting with all subsequences of length 2, to length L:

$$\boldsymbol{\gamma(i, j) = \max} \begin{cases} \boldsymbol{\gamma(i + 1, j)}, \\ \boldsymbol{\gamma(i, j - 1)}, \\ \boldsymbol{\gamma(i + 1, j - 1) + \delta(i, j)}, \\ \boldsymbol{\max_{i<k<j} [\gamma(i, k) + \gamma(k + 1, j)]} \end{cases}$$

The fill stage is $O(L^2)$ in memory and $O(L^3)$ in time [1].

4.1.3 Traceback Stage

The traceback stage makes use of a stack construct in order to handle the case in which there is a bifurcation. When a bifurcation happens two optimal substructures are joined together and the stack is used to travel down one structure and then pop back so it can travel down the second structure. The traceback stage works by first pushing the element in array position (1, L) onto the stack. This element represents the maximum number of base pairs in the optimally aligned structure. It is possible that there are other positions in the array with the same number of base pairs as the optimally aligned structure. The algorithm then traces back down the array as described below [1].

Initialization: Push (1, L) onto stack.
Recursion: Repeat until stack is empty:

- pop **(i, j)**.
- if **i** >= **j** continue;
  else if **γ(i + 1, j) = γ(i, j)** push **(i + 1, j)**;
  else if **γ(i, j – 1) = γ(i, j)** push **(i, j – 1)**;
  else if **γ(i + 1, j – 1) + $\delta_{i,j}$ = γ(i, j)**:
    - record **i, j** base pair.
    - push **(i + 1, j – 1)**.
  else for **k = i + 1 to j – 1: if γ(i, k) + γ(k + 1, j) = γ(i, j)**:
    - push **(k + 1, j)**.
    - push **(i, k)**.
    - break.

The traceback stage is linear in memory and time [1].

4.1.4 Enhancements to Algorithm

There are several enhancements which can be made to the Nussinov algorithm. First, the formal definition of the algorithm only allows for Watson-Crick pairs. Second, both A-U and C-G pairs are given the same value in the scoring matrix. It might be desirable to allow C-G pairs to have a higher value since the C-G bonds are slightly stronger because of three hydrogen bonds instead of two for A-U. There is also a case in nature where G-U bonds occur. It might be desirable to allow these so called wobble pairs to occur in the algorithm. With just a minor modification to the scoring function both of the cases can be accounted for.

Another enhancement to the algorithm would be to put a limit on how small a hairpin loop can be. There is a biological limit on the minimum length of hairpin loop to 3 bases. The algorithm allows for hairpin loops of lengths less then three, so another minor modification could be made to limit the hairpin loops to three bases.

4.2 Nusssinov SCFG Version

The stochastic context free grammar version of the Nussinov algorithm is a probabilistic algorithm. A grammar is first setup which has all the production rules for the grammar which includes a single non-terminal **(See Figure 4.1)**. Probabilities are then assigned to each production rule based on any one of several methods. Probabilities could be assigned by counting state transitions in known RNA and then converting the counts to probabilities [1]. Another method is to use an Expectation Minimization algorithm to calculate the probabilities [32]. Finally, the probabilities could be gained from subjective estimation [1].

S -> aS | cS | gS | uS	(i unpaired)
S -> Sa | Sc | Sg | Su	(j unpaired)
S -> aSu | cSg | gSc | uSa	(i, j pair)
S -> SS	(bifurcation)

Figure 4.1. Production Rules [1]

4.2.1 Formal Algorithm Definition

The SCFG version of the Nussinov algorithm actually uses a slightly modified CYK (Cocke – Younger – Kasami) parsing algorithm. Typically CYK parsing algorithms are used on grammars that in Chomsky normal form but by slightly modifying the CYK algorithm a more efficient approach can be used [1].

The algorithm starts out by letting the probability parameters for the grammar's productions rules be denoted by p(aS), p(cS), p(gS), p(uS), p(Sa), p(Sc), p(Sg), p(Su), p(aSU), p(cSg), p(gSc), p(uSa), and p(SS). Next the fill stage begins as follows [1]:

4.2.2 Fill Stage

The SCFG version of the Nussinov algorithm starts out similar to the standard Nussinov algorithm by creating a rectangular array for the dynamic programming matrix. The diagonal of the array is initialized to be negative infinity. The lower diagonal is initialized to the maximum of the probability values for the character at the given position through either the i-unpaired or j-unpaired rule. Then the recursion phase begins and the algorithm travels down the diagonals calculating the probabilities based on the algorithm.

Initialization:

$$\gamma(i, i-1) = -\infty \text{ for } i = 2 \text{ to } L;$$

$$\gamma(i, i) = \max \begin{cases} \log p(x_i S) \\ \log p(S x_i) \end{cases} \quad \text{For } i = 1 \text{ to } L$$

Recursion: for i = 1 to L – 1, j = i + 1 to L:

$$\gamma(i, j) = \max \begin{cases} \gamma(i+1, j) + \log p(x_i S); \\ \gamma(i, j-1) + \log p(S x_j); \\ \gamma(i+1, j-1) + \log p(x_i S x_j); \\ \max_{i<k<j} \gamma(i, k) + \gamma(k+1, j) + \log p(SS) \end{cases}$$

4.2.3 Traceback Stage

The traceback stage of the SCFG version of the Nussinov algorithm works almost exactly like the standard version of the algorithm. The traceback starts at the array position (1, L) pushes that onto the stack and then recursively traceback through the array until the stack is empty. The main difference between this traceback and the standard traceback is that instead of using the scoring matrix during traceback the algorithm instead uses the probability values.

Now that the Nussinov algorithm has been detailed as well as all known methods to visualize the RNA secondary structure, all these components can be joined together to describe the visualization interface. The next chapter will show how the Nussinov algorithm is used to calculate the secondary structure and display the secondary structure in several ways simultaneously. The next chapter will also detail the way in which the

visualization will tie information from two different representations together through the use of three dimensional lines linking significant information.

Chapter 5

The Visualization Interface

This chapter gives a high level description of the interface used by the visualization. It starts out by describing which data representations will be used by the visualization. It then shows how the visualization uses a three dimensional stage model to draw the data representations onto facets of the stage. Some of the visualizations unique aspects are described in this chapter. The chapter describes how the visualization can step through an RNA sequence and display intermediate values in the calculation. It is also shown how the visualization makes use of three dimensional lines in order to draw connections between pertinent information in the Nussinov calculation.

5.1 Visualization Interface

The interface to the visualization is composed of a three dimensional space which is used to display the results of the Nussinov algorithm. The main component in the three dimensional space is an object referred to as the stage. The stage is actually a three dimensional model which resembles a theatrical stage. The stage is composed of several angular faces which are referred to as facets **(See Figure 5.1)**. Each facet contains a unique representation of the Nussinov algorithm calculation. For example the back facet, referred to as the planar graph facet, contains the two dimensional representation of the secondary structure of the RNA sequence. The facet on the back right, referred to as the circular facet, contains the circular graph representation of the secondary structure. The

back left facet, referred to as the bracketed facet, contains the bracketed representation. Finally, the bottom facet, referred to as the matrix facet, contains the matrix representation of the dynamic programming matrix. The stage has been constructed in such a way as to convey the maximum amount of information while at the same time maintaining a clean interface which is not cluttered with to much information.

There are two unique aspects to this interface. First, the interface makes use of three dimensional space in order to display multiple two dimensional representations of data. Most other secondary structure prediction programs only allow the data to be represented one way at a time. By using three dimensional space and by displaying more information it is intended that the user will be able to have a more intuitive understanding and will be able to draw more connections and infer information from the visualization. The second unique aspect of the visualization is that it allows the user to step through the RNA sequence as the secondary structure is being calculated and displayed.

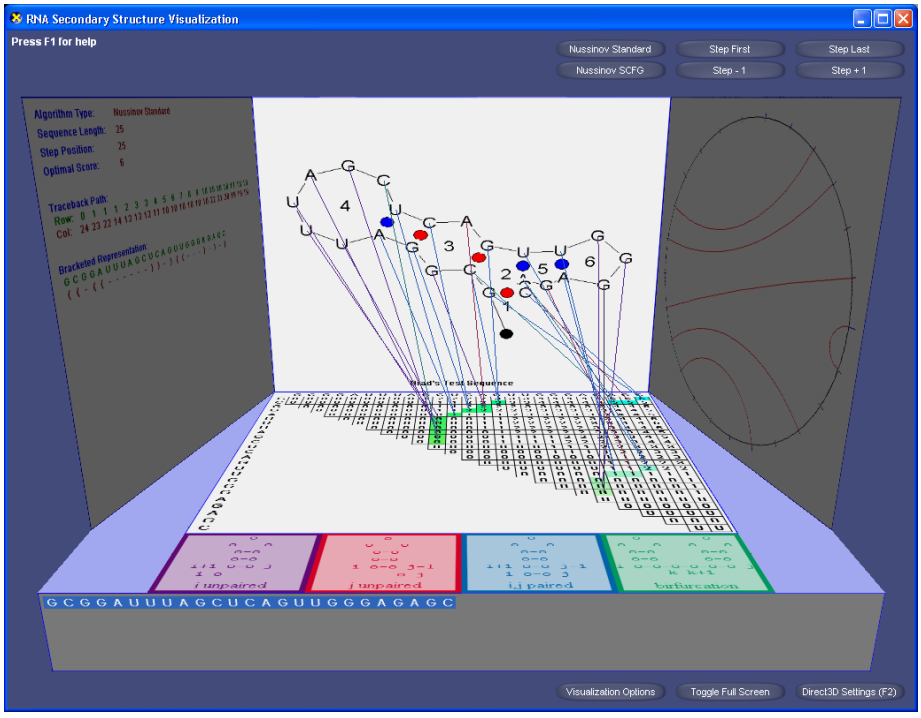


Figure 5.1. The Stage Object

5.2 Stepping Through the Algorithm

The visualization program allows the user to enter any given RNA sequence and then step through the sequence. In order to facilitate the stepping features, the visualization contains four buttons located on the main interface **(See Figure 5.1)**. These buttons are labeled ***Step First***, ***Step – 1***, ***Step + 1*** and ***Step Last***. The stepping feature works as follows: if an RNA sequence consisting of 25 bases has been entered into the visualization, then after entering the sequence the visualization will automatically step to the last position to show the complete secondary structure of the sequence. The user can then click on any of the four stepping buttons. By clicking the ***Step First*** button the visualization will move to the first base in the sequence and all the facets will be recalculated based on the first nucleotide base in the sequence. If the ***Step + 1*** button were now clicked then the next base in the sequence would be included and all the facets would again be recalculated. The user could continue to step through the whole sequence in this manner.

5.3 Three Dimensional Lines

Another unique feature of the visualization is that it takes advantage of the three dimensional space in order to draw connections between the stage facets. The visualization will draw three dimensional lines between the planar graph facet and the matrix facet in order to show where each nucleotide in the planar graph originated from in the dynamic programming matrix. In addition, the color of the three dimensional lines corresponds to the Nussinov structure type that produced the nucleotide in the algorithm. There are four color coded boxes in front of the matrix graph facet **(See Figure 5.1)**. The purple box represents the i-unpaired structure, the red box represents the j-unpaired structure, the blue box represents the i,j-paired structure and the green box represents a bifurcation. The lines are color coded the same color as the boxes to show which structure was used to calculate the current position.

This chapter gave an overview of the visualization interface so now the next chapter will detail how the stage object is constructed. The next chapter will detail how the Microsoft DirectX API is used to create a three dimensional visualization space.

Chapter 6

Three Dimensional Stage

This chapter describes how the Microsoft DirectX API has been used to create the three dimensional stage object of the visualization. The chapter starts by giving a brief overview of the DirectX API, and then describes how some of the DirectX constructs are used to create the stage object. The chapter details the triangle strips that were created to complete the stage object. It also details the internal dimensions of the stage object. Next, it is shown how an index buffer is used in order to increase the performance of the drawing functions. Finally, the chapter details how the DirectX texturing mechanism is used to draw images onto the stage object.

6.1 Three Dimensional Stage

The three dimensional stage is the structure that the visualization uses as a canvas to paint its multiple data representations upon. The stage is actually a three dimensional model composed of polygons that with the help of Microsoft's DirectX 9.0 software development kit, are rendered to the screen. The DirectX SDK is a collection of low level application programming interfaces that are used for creating high performance 2D and 3D graphics. The DirectX SDK consists of three main API interfaces, Direct3D, DirectInput, and DirectSound **[27]**. Since this visualization is both highly graphical and the visualization is utilizing a three dimensional view space, both of which are cpu

intensive, performance was a critical consideration. Based on this consideration the Direct3D interface offered by DirectX was a perfect fit for the visualization.

6.2 Direct3D And The Three Dimensional Stage

Direct3D is the application programming interface used for displaying three dimensional objects. The main advantage of Direct3D is that it utilizes whichever graphics accelerator device is installed on the machine without requiring the programmer to write any code specific to the particular brand of graphics accelerator. The Direct3D API provides a set of functions which all graphics accelerator manufacturers support.

In order to create a three dimensional stage object using Direct3D, the API provides several types of primitive objects which can be utilized. The Direct3D primitives include a Point List, Line List, Line Strip, Triangle List, Triangle Strip, and a Triangle Fan. In addition each primitive consists of one or more vertices which define the endpoints of the primitive type. The Direct3D API provides a flexible mechanism for defining the vertices depending on the requirements of the application. For example, if the application only requires the three dimensional position of a vertex without color, lighting or texturing then you could define a vertex as follows:

```
#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ)
struct CUSTOMVERTEX
{
   float x,y,z; // Position of vertex in 3D space
};
```

Listing 6.1. Custom Vertex (Position Only)

Alternately, if a vertex requires the three dimensional position as well as color then the vertex could be defined as follows:

```
#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ | D3DFVF_DIFFUSE)
struct CUSTOMVERTEX
{
   float x,y,z; // Position of vertex in 3D space
   DWORD color; // Diffuse color of vertex
};
```

Listing 6.2. Custom Vertex (Position and Color)

The types of vertices required by the visualization program are first vertices used for the stage object and second vertices used for the lines which are drawn to connect different facets of the stage. The stage vertices need to have a three dimensional position as well as normal vectors for lighting calculations as well as texture coordinates so that the visualization output can be drawn onto the stage. A stage vertex is defined as follows:

```
#define STAGE_D3DFVF_CUSTOMVERTEX(D3DFVF_XYZ|D3DFVF_NORMAL|D3DFVF_TEX1)
struct STAGE_CUSTOMVERTEX
{
   float x, y, z;    //Position of vertex in 3D space
   float nx, ny, nz; //Normal vector for lighting calculations
   float tu, tv;     //Texture coordinates
};
```

Listing 6.3. Custom Vertex used by stage object

The vertices for the three dimensional lines need to have a three dimensional position as well as normal vectors for lighting as well as a color component. A line vertex is defined as follows:

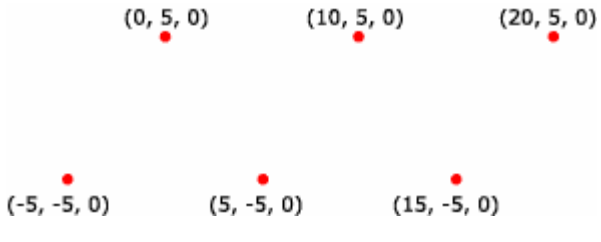
```
#define LINE_D3DFVF_CUSTOMVERTEX(D3DFVF_XYZ|D3DFVF_NORMAL|D3DFVF_DIFFUSE)
struct LINE_CUSTOMVERTEX
{
   float x, y, z;    // Position of vertex in 3D space
   float nx, ny, nz; // Normal vector for lighting calculations
   DWORD color;      // Diffuse color of vertex
};
```

Listing 6.4. Custom Vertex used by visualization lines

Once the vertices have been defined then the vertices can be used to create the primitive types used by Direct3D. The simplest of the primitive types is the point list. In order to draw a set of points using Direct3D one would first specify an array of vertices with the required vertex attributes as described above. Once the array of vertices was created then the Direct3D API function used from drawing primitive types would be called. The code for drawing this point list would be as follows **[27]**:

```
struct CUSTOMVERTEX
{
   float x,y,z;
};

CUSTOMVERTEX Vertices[] =
{
   {-5.0, -5.0, 0.0},
```



```
   { 0.0,  5.0,  0.0},
   { 5.0,  -5.0,  0.0},
   {10.0,  5.0,  0.0},
   {15.0,  -5.0,  0.0},
   {20.0,  5.0,  0.0}
};

d3dDevice->DrawPrimitive(D3DPT_POINTLIST, 0, 6);
```

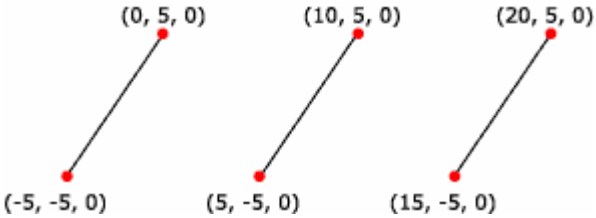
Point List **[27]**

Listing 6.5. Point List

The code for drawing a line list would be as follows **[27]**:

```
struct CUSTOMVERTEX
{
   float x,y,z;
};

CUSTOMVERTEX Vertices[] =
{
   {-5.0,  -5.0,  0.0},
   { 0.0,   5.0,  0.0},
   { 5.0,  -5.0,  0.0},
   {10.0,   5.0,  0.0},
   {15.0,  -5.0,  0.0},
   {20.0,   5.0,  0.0}
};
d3dDevice->DrawPrimitive(D3DPT_LINELIST, 0, 3);
```



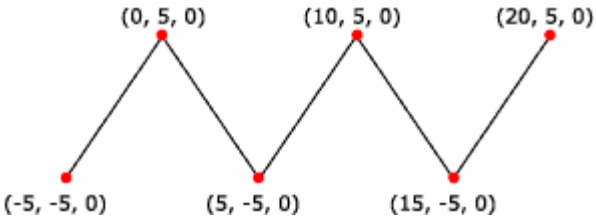
Line List **[27]**

Listing 6.6. Line List

The code for drawing a line strip would be as follows **[27]**:

```
struct CUSTOMVERTEX
{
   float x,y,z;
};

CUSTOMVERTEX Vertices[] =
{
   {-5.0,  -5.0,  0.0},
   { 0.0,   5.0,  0.0},
   { 5.0,  -5.0,  0.0},
   {10.0,   5.0,  0.0},
   {15.0,  -5.0,  0.0},
   {20.0,   5.0,  0.0}
};
d3dDevice->DrawPrimitive(D3DPT_LINESTRIP, 0, 5);
```



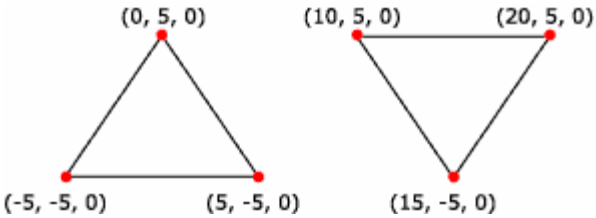
Line Strip **[27]**

Listing 6.7. Line Strip

The code for drawing a triangle list would be as follows **[27]**:

```
struct CUSTOMVERTEX
{
   float x,y,z;
};

CUSTOMVERTEX Vertices[] =
{
   {-5.0, -5.0, 0.0},
   { 0.0,  5.0, 0.0},
   { 5.0, -5.0, 0.0},
   {10.0,  5.0, 0.0},
   {15.0, -5.0, 0.0},
   {20.0,  5.0, 0.0}
};
d3dDevice->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 2);
```



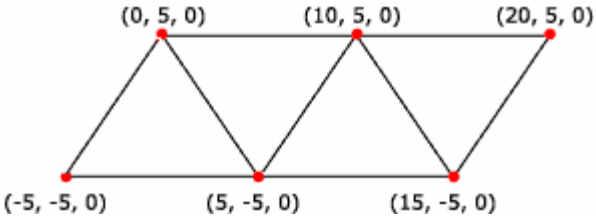
Triangle List **[27]**

Listing 6.8. Triangle List

The code for drawing a triangle strip would be as follows **[27]**:

```
struct CUSTOMVERTEX
{
   float x,y,z;
};

CUSTOMVERTEX Vertices[] =
{
   {-5.0, -5.0, 0.0},
   { 0.0,  5.0, 0.0},
   { 5.0, -5.0, 0.0},
   {10.0,  5.0, 0.0},
   {15.0, -5.0, 0.0},
   {20.0,  5.0, 0.0}
};
d3dDevice->DrawPrimitive(D3DPT_TRIANGLESTRIP, 0, 4);
```



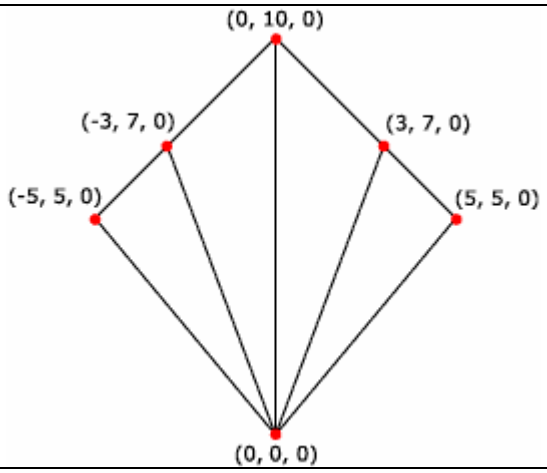
Triangle Strip **[27]**

Listing 6.9. Triangle Strip

The code for drawing a triangle fan would be as follows **[27]**:

```
struct CUSTOMVERTEX
{
   float x,y,z;
};

CUSTOMVERTEX Vertices[] =
{
   { 0.0, 0.0, 0.0},
   {-5.0, 5.0, 0.0},
   {-3.0,  7.0, 0.0},
   { 0.0, 10.0, 0.0},
   { 3.0,  7.0, 0.0},
   { 5.0,  5.0, 0.0},
};
```



	Triangle Fan **[27]**

```
d3dDevice->DrawPrimitive(D3DPT_TRIANGLEFAN, 0, 4);
```

Listing 6.10. Triangle Fan

6.2.1 Stage Triangle Strips

The three dimension stage object is composed of triangle strips as shown above. In order to create the stage object, eight triangle strips were used. The first triangle strip defines the front panel where the progress bar will be placed. The second triangle strip defines the region of the stage where the four Nussinov structure types will be placed. The third and fourth triangle strips are some unused area but are necessary to connect the structure together. The fifth and seventh triangle strips are also unused sections. The sixth triangle strip is the panel where the matrix graph will be placed. The eighth triangle strip has sections for the bracketed graph, planar graph and circular graph panels **(See Figure 6.1)**.

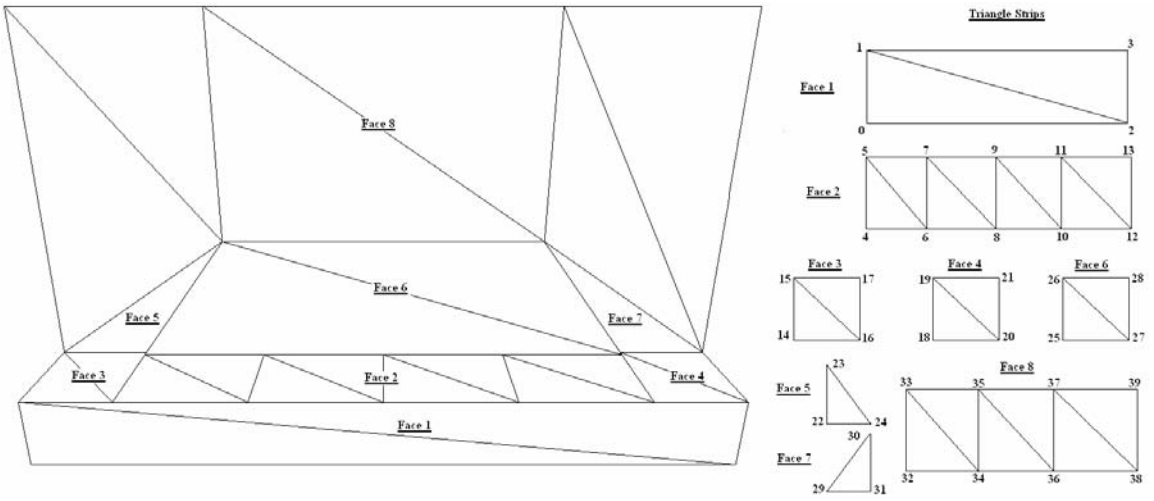


Figure 6.1. Stage model showing triangle strips

6.2.2 Stage Dimensions

Now that the triangle strips have been defined the next step is to create dimensions for the stage object and to define the vertices positions. The stage object will have its own coordinate system so the dimensions are made relative to the size of the facets needed in the stage object. The facets which will contain the matrix graph, bracketed graph, planar

graph and circular graph will all be the same size so define those facets as 10 units by 10 units. The four facets which will contain the four Nussinov structure types will be the same with as one of the 10 by 10 facets so each of those facets has to be 10 / 4 = 2.5 units. The progress bar facet is 22 units wide by 2 units high. When completely put together the dimensions turn out as follows: **(See Figure 6.2)**

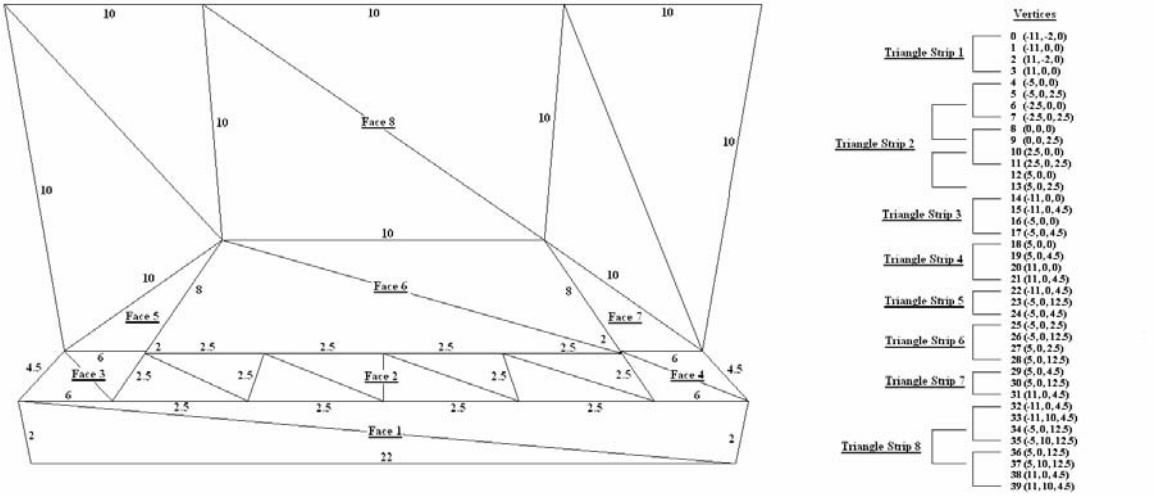


Figure 6.2. Stage model show dimensions and vertices

6.2.3 Direct3D Index Buffer

Drict3D offers a structure called an index buffer which helps to increase the performance of drawing the model by combining vertices which overlap. When the stage object is put together and the triangle strips are arranged next to each other there are some vertices with the same coordinates. For example vertex 1 in triangle strip 1 is at the same coordinate as vertex 14 in triangle strip 3. This in fact happens many times in the model so the index buffer is used as an integer offset into the vertex buffer. This buffered offset increases the performance of the drawing functions **[27]**. The index buffer for the stage object would be setup as follows:

```
//Index Buffer Values
WORD pIndices[] = { 0, 1, 2, 3, 2, 1,      //Polygon 0
                    4, 5, 6, 7, 6, 5,      //Polygon 1
                    8, 9,10,11,10, 9,      //Polygon 2
                   12,13,14,15,14,13,      //Polygon 3
                   16,17,18,19,18,17,      //Polygon 4
                   20,21,22,23,22,21,      //Polygon 5
```

```
24,25,26,27,26,25,      //Polygon 6
28,29,30,               //Polygon 7
31,32,33,34,33,32,      //Polygon 8
35,36,37,               //Polygon 9
38,39,40,41,40,39,      //Polygon 10
42,43,44,45,44,43,      //Polygon 11
46,47,48,49,48,47};     //Polygon 12
```

6.3 Texture And Three Dimensional Stage

Now that it has been shown how to create the three dimensional stage, the next operation would be to draw something onto the stage. The process of drawing a picture onto a polygon is called texturing the polygon. A texture is simply an image file such as a bitmap or jpeg. The picture is arranged in such a way that portions of the picture can be extracted from the image file and connected to vertices in the model. The texture image is a two dimensional picture where the x, y coordinates are represented by the letters u and v. The dimensions of the texture image are from 0 to 1 so the upper left corner of a texture image would be at coordinate (0, 0) and the lower right coordinate would be (1, 1) **(See Figure 6.3)**.

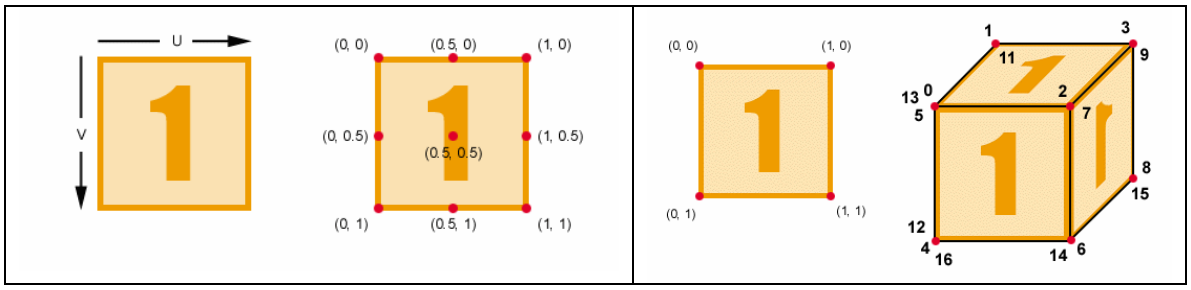


Figure 6.3. Texture Coordinates [28]

This visualization program uses the Direct3D texturing mechanism to draw the data representations on the stage. The texture file used for the texture is dynamically calculated by the program and then mapped onto the stage object. Recall that each vertex in the stage object has several components. The vertex contains the x, y, z position of the vertex as well as the nx, ny, nz normal vector for lighting. The last component that the vertex contains in the tu, tv texture coordinates **(See Listing 6.3)**. The mapping works by connecting the tu, tv texture coordinate of the vertex to the correct location within the texture image **(See Figure 6.4)**.

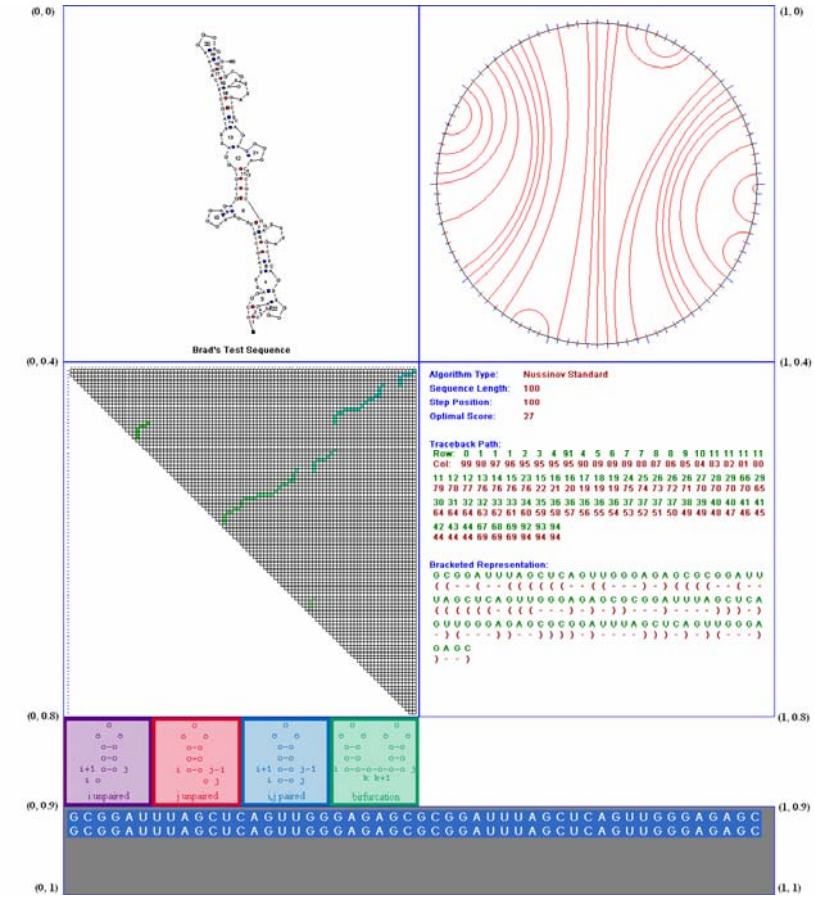


Figure 6.4. Visualization Texture

This chapter detailed how the stage object was constructed and displayed through the help of Microsoft DirectX. The chapter ended showing how the texturing functionality of DirectX could be used to draw an image onto the stage object. The next chapter will explain how to create the image that is used as the texture. There are actually several calculations which make up the components of the image so each one will be detailed individually.

Chapter 7

Texture Construction

The secondary RNA structure visualization has five main facets. The five facets are the planar graph facet, the circular graph facet, the matrix graph facet, the bracketed graph facet and the progress bar facet. This chapter goes into detail on how each of these facets are constructed. Each of the facets share a common coordinate system so this chapter first details how the coordinate system is setup and then each facet has a brief overview before diving into the detail of the construction. Each facet is divided into a subsection which first details the input to the facet calculation, then the actual calculation is detailed and finally the final result is shown.

7.1 Coordinate System Setup

The coordinate system used by Microsoft Windows is a Cartesian coordinate system where the origin, position (0, 0), is positioned at the upper left hand corner of the screen. The X axis moves in the positive direction as you move to the right across the screen and the Y axis moves in the positive direction as you move down the screen **(See Figure 7.1)**. All calls to Windows GDI drawing functions would use this coordinate system, with the origin at the upper left corner of the screen, as the basis for their drawing. This coordinate setup may not be ideal for all applications. For example you might want the origin to be at the center of the screen and you might want the Y axis to move in the positive direction as it moves up instead of moving down like it does. Luckily, this is just

the default setup for the coordinate system and it can easily be changed. Each of the main facets in this visualization are setup as 512 x 512 pixel bitmaps. The origin of the coordinate system is moved so that it is positioned at the exact center of the bitmap drawing area. The Y axis is also reconfigured so that the positive Y axis points up as would be expected in a standard Cartesian coordinate setup.

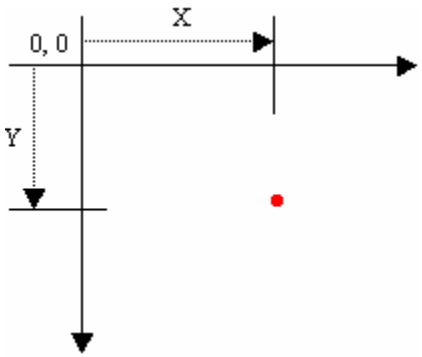


Figure 7.1. Windows Coordinate System [23]

The Win32 API provides a rich set of functions that can be used to manipulate the coordinate system. To change the position of the origin so that it is at the center of the 512 x 512 drawing area of the facet, a call to the **SetViewportOrgEx** function is used **(See Listing 7.1)**. The SetViewportOrgEx function is used to reposition the origin of the coordinate system. The function takes four parameters, a handle to the current drawing device context, the x-coordinate origin position, the y-coordinate origin position and a pointer to the original origin position. After this function call the coordinate system is setup where the origin is position as desired but the Y axis is still upside down **(See figure 7.2)**.

```
SetViewportOrgEx(hdc, (int)(fTextureSize / 2.0f), (int)(fTextureSize / 2.0f), NULL);
```

```
BOOL SetViewportOrgEx(
  HDC hdc,          // handle to device context
  int X,            // new x-coordinate of viewport origin
  int Y,            // new y-coordinate of viewport origin
  LPPOINT lpPoint // original viewport origin
);
```

Listing 7.1. SetViewPortOrgEx() Function (Windows API) [24]

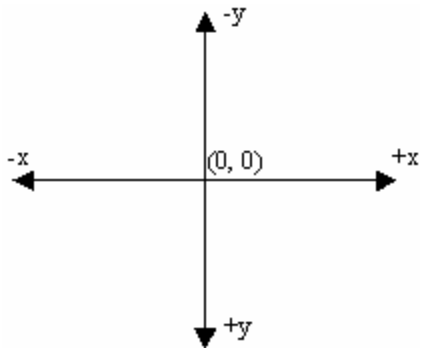


Figure 7.2. Coordinate With Modified Origin [23]

To change the direction of the Y axis and setup the coordinate system as desired several more Win32 API functions are need. First a call to the **SetMapMode** function **(See Listing 7.2)** is needed in order to specify the coordinate systems unit of measure. The SetMapMode function has two parameters. The first parameter is a handle to the current drawing surface device context and the second parameter is the desired mapping mode. The images drawn for each of the facets will use the **MM_ISOTROPIC** mapping mode which means that one unit in the horizontal axis is equivalent to one unit in the vertical axis. The last step is to set the scaling factor between the window and the viewport. To set the horizontal and vertical extents of the window the **SetWindowExtEx** function **(See Listing 7.2)** is used. This function takes four parameters, a handle to the current drawing surface device context, the horizontal window extent, the vertical window extend and a pointer to the original window extents. To set the horizontal and vertical extents of the viewport the **SetViewPortExtEx** function **(See Listing 7.2)** is used. This function works in exactly the same way as the **SetWindowExtEx** function except that it modifies the viewport extents. The interesting thing to notice is that the vertical viewport extent is set to -1. This has the affect of reversing the direction of the Y axis so that it points in the direction as desired **(See Figure 7.3)**.

```
SetMapMode(hdc, MM_ISOTROPIC);
SetWindowExtEx(hdc, 1, 1, NULL);
SetViewportExtEx(hdc, 1, -1, NULL);
```

```
int SetMapMode(
  HDC hdc,        // handle to device context
  int fnMapMode   // new mapping mode
);

BOOL SetWindowExtEx(
```

```
  HDC hdc,        // handle to device context
  int nXExtent,   // new horizontal window extent
  int nYExtent,   // new vertical window extent
  LPSIZE lpSize   // original window extent
);

BOOL SetViewportExtEx(
  HDC hdc,        // handle to device context
  int nXExtent,   // new horizontal viewport extent
  int nYExtent,   // new vertical viewport extent
  LPSIZE lpSize   // original viewport extent
);
```

Listing 7.2 Windows API Functions [24]

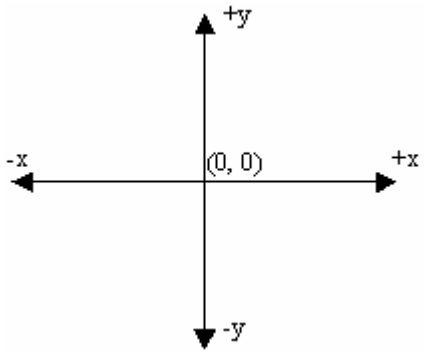


Figure 7.3.Final Coordinate System Configuration [23]

7.2 Circular Graph Calculation

The circular facet as described in section 3.5.6, has several components to it. First, there are small blue tick marks on the outer edge of the circle that help identify the position of the RNA nucleotides in the circular representation. Every 10^{th} tick mark is slightly longer so that when long sequences are visualized there is an easy division that helps keep track of the current position in the overall sequence. Second, there are small red tick marks on the inner edge of the circle that represent nucleotides which do not have bonds. Lastly, there are red arcs on the inner surface of the circle that connected nucleotides that have a bond in the secondary structure. All of these elements will be drawn onto a 512 x 512 bitmap using windows GDI (graphical device interface) commands.

7.2.1 Circular Graph Calculation Input

The input to the circular facet calculation is the output from the Nussinov algorithm which is an array that represents the secondary structure in bracketed form. For example given the RNA sequence GGGAAAUCC the output from the Nussinov algorithm would have been an array of nine elements in this form:

```
Position     0  1  2  3  4  5  6  7  8
Sequence     G  G  G  A  A  A  U  C  C
Nussinov = [ -, (, (, -, -, (, ), ), ) ]
```

The circular facet actually uses a slight variation on the array from above. Instead of the array being a character array where each element in the array holds a dash or a bracket the actual input to the circular calculation uses an integer array where dashes are replaced with the positions array index and each bracket position holds the index to the matching bracket. For example, the bracket at array position 1 matches with the bracket at array position 8 so the new array will store 8 in array index 1. When all brackets are replaced with their matching index values then the following array is obtained:

```
Position     0  1  2  3  4  5  6  7  8
Nussinov = [ 0, 8, 7, 3, 4, 6, 5, 2, 1 ]
```

7.2.2 Bounding Box Dimensions

The circular representation is going to be drawn onto a bitmap surface that is 512 x 512 pixels. There will be a border around the bitmap which will take 5 pixels from each side of the bitmap. Additionally, there will be 20 pixels of padding on the interior of the border. Let **fTextureSize** = **512.0f**, **fBorder** = **5.0f** and **fPadding** = **20.0f**. Therefore, the radius of the circle will be:

fRadius = (**fTextureSize** / **2.0f**) – **fBorder** – **fPadding**;
fRadius = **231.0f**

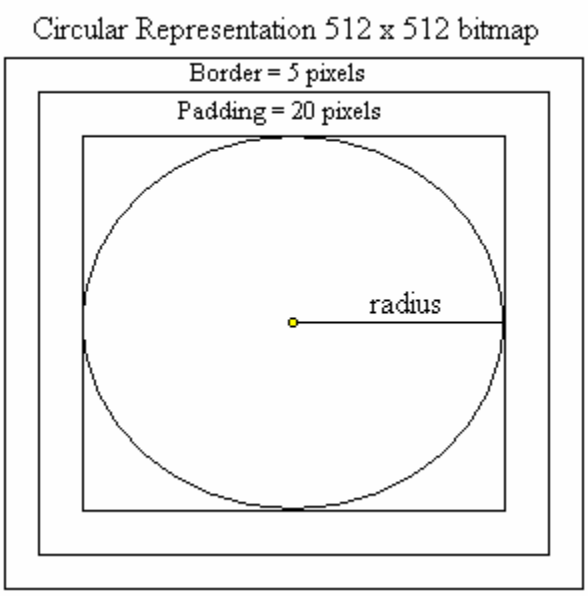


Figure 7.4. Facet Bounding Boxes

7.2.3 Calculating Tick Marks

In order to draw the tick marks around the circular representation the first thing that needs to be done is to determine where on the circle the tick marks should be drawn. Given that the input array contains a sequence of length **x**, (**x = 9** in this example), the circle must be segmented into **x** equal pie pieces. Since the circle is 360 degrees then dividing 360 by **x** tells us the angle between each item on the circle. In the example sequence **x = 9** so **360 / 9 = 40** degrees between each tick mark. There are two types of tick marks each of which have a slightly different length. The smaller tick marks will be drawn on the exterior of the circle and they will be aligned with nucleotide positions. The longer tick marks will be drawn every 10^{th} tick mark to give a better indication of the position in the sequence. Since both the smaller and larger tick marks are drawn on the exterior of the circle then two imaginary circles will be used to draw the tick marks. The first imaginary circle will have radius 5 pixels larger than the main circle and the second imaginary circle will have a radius that is 10 pixels larger. The smaller tick marks will be created by drawing lines from positions on the circle with radius + 5 to the center of the circle and the larger tick marks will be created by drawing lines from the circle with radius + 10 to the center of the circle. The lines are drawn using the windows GDI **MoveToEx** and **LineTo** functions **(See Listing 7.3)**.

```
BOOL MoveToEx(
HDC hdc,          // handle to device context
int X,            // x-coordinate of new current position
int Y,            // y-coordinate of new current position
LPPOINT lpPoint   // old current position
);
BOOL LineTo(
HDC hdc,    // device context handle
int nXEnd,  // x-coordinate of ending point
int nYEnd   // y-coordinate of ending point
);
```

Listing 7.3 Windows API Functions [24]

Since we know the angle between each position on the circle and we know the coordinate of the center of the circle then we need to calculate the coordinate of the starting point of each line. The program loops through the length of the sequence, at each position the angle to the position is calculated and then the coordinate of the starting point of the line is calculated using the following formula: **(See Figure 7.5)**

Center of circle = (px, py)
$x = r \cos(\theta) + px$
$y = r \sin(\theta) + py$

The program checks to see if it is on an increment of 10 by checking if the modulus of the loop increment equals 0. If it is then the longer tick marks are drawn.

```
// Draw the tick marks around the circle
hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 255));
hOldPen = (HPEN)SelectObject(hdc, hPen);
for(int x = 0; x < iStep; x++)
{
   if((x % 10) == 0) // Draw longer tick every 10th position
   {
      xPos = (int)(((fRadius + 10.0f) * cos(x * fAngle)) + fCenterX)
      yPos = (int)(((fRadius + 10.0f) * sin(x * fAngle)) + fCenterY)
      MoveToEx(hdc, xPos, yPos, NULL);
   }
   Else // Draw shorter tick
   {
      xPos = (int)(((fRadius + 5.0f) * cos(x * fAngle)) + fCenterX)
      yPos = (int)(((fRadius + 5.0f) * sin(x * fAngle)) + fCenterY)
      MoveToEx(hdc, xPos, yPos, NULL);
   }
   LineTo(hdc, (int)fCenterX, (int)fCenterY);
}
SelectObject(hdc, hOldPen);
DeleteObject(hPen);
```

Once all the lines have been drawn then the main circle will be drawn with the interior filled with the background color. This will both draw the circle and mask the interior of the lines.

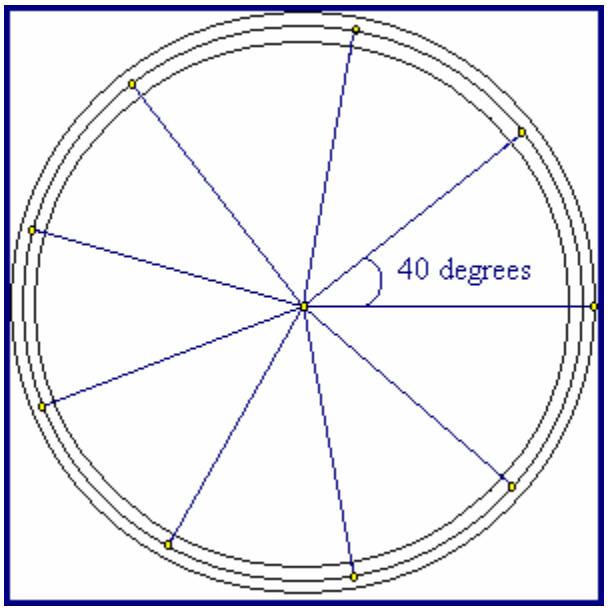


Figure 7.5. Circular Facet (Degrees between positions)

7.2.4 Calculating Arcs

The first piece of information needed in the calculation used to draw the arcs onto the circle is to know which RNA nucleotides will be paired so that arcs can be drawn between these pairs. This information can be determined from the input array. The input array was setup so that each element in the array either holds the index of the matching pair or it holds its linear position if no matching pair exists.

```
Position     0  1  2  3  4  5  6  7  8
Nussinov = [ 0, 8, 7, 3, 4, 6, 5, 2, 1 ]
```

To determine which positions need arcs to be drawn, a loop is setup on the array index. While looping through the array if the element at the current index equals the index then no arc is necessary. If the element at the current index contains a value greater than the current index then an arc is necessary. Notice in the array above that the position of the arc can be determined at two places. For example, at array index 1 the value is 8. That

means that an arc is necessary from position 1 to position 8. But this information can also be determined from array index 8 which has a value of 1. By convention the arcs will always be drawn when the trailing end of the arc has been encountered. This can be determined by checking to see if the value of the element at the current index is less than the index. If true then the trailing side of the arc has been encountered and the arc should be drawn. The arcs will be drawn with the windows GDI **Arc** function **(See Listing 7.4)**.

```
BOOL Arc(
  HDC hdc,          // handle to device context
  int nLeftRect,    // x-coord of rectangle's upper-left corner
  int nTopRect,     // y-coord of rectangle's upper-left corner
  int nRightRect,   // x-coord of rectangle's lower-right corner
  int nBottomRect, // y-coord of rectangle's lower-right corner
  int nXStartArc,   // x-coord of first radial ending point
  int nYStartArc,   // y-coord of first radial ending point
  int nXEndArc,     // x-coord of second radial ending point
  int nYEndArc      // y-coord of second radial ending point
);
```

Listing 7.4. Windows API Arc() Function [24]

If it has been determined that an arc should be drawn from array index **x** to index **y** then the first thing to do is to calculate where the corresponding points should be on the circle. Let the angle between the positions on the circle be represented by **θ**. Let **θ1** be the angle to the first point which can be calculated by **θ1 = x * θ**. Let **θ2** be the angle to the second point which can be calculated by **θ2 = y * θ**. Now that the angles to the positions on the circle can be determined, next the coordinate positions of the start and end points of the arc need to be determined. Let **P1** be the start point and **P2** be the end point. Let **P0** be the center of the circle. Points **P1** and **P2** and be calculated as follows: **(See Figure 7.6)**

P1 = (r cos(θ1) + P0x, r sin(θ1) + P0y)
P2 = (r cos(θ2) + P0x, r sin(θ2) + P0y)

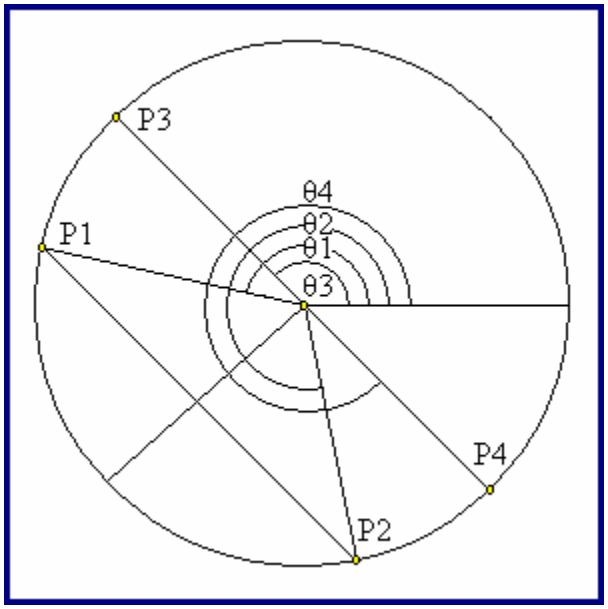


Figure 7.6. Circular Facet (Intermediate Calculation)

Now that the start and end points of the arc have been calculated to be **P1** and **P2**, the next calculation is to determine the apex of the arc. The first step in this calculation is to find point **P3** and **P4** which are the end points of a chord passing through the center of the circle and parallel to the chord **P1**, **P2 (See Figure 7.6)**. The angle to point **P3** can be calculated by finding the angle half way between **θ1** and **θ2** and subtracting 90 degrees from that angle. Let **θx** be the angle half way between **θ1** and **θ2**.

θx = θ1 + ((θ2 - θ1) / 2)
θ3 = θx – 90 degrees
P3 = (r cos(θ3) + P0x, r sin(θ3) + P0y)
θ4 = θx + 90 degrees
P4 = (r cos(θ4) + P0x, r sin(θ4) + P0y)

Now that point **P1**, **P2**, **P3** and **P4** are known it's time to find the position of the apex of the arc. The position is calculated by drawing lines from **P1** to **P4** and from **P2** to **P3**. The intersection of these lines will be the apex of the arc **(See Figure 7.7)**.

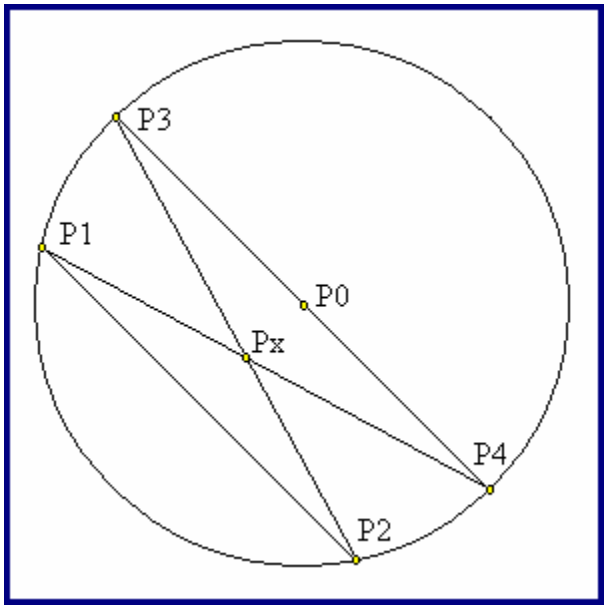


Figure 7.7. Circular Facet (Arc Apex Determination)

Let the position of the intersection of the lines be represented by the point **Px**. To calculate the position of **Px** it is necessary to find the equation for line **P1P4** and **P2P3** and then to solve two equations and two unknowns to find **Px**. First determine the equation of lines **P1P4** and **P2P3** in slope intercept form:

$\mathbf{y = m_1x + b_1}$
$\mathbf{y = m_2x + b_2}$

To calculate the slope and the y intercept use the equation of slope and the intercept equation:

m = (y2 – y1) / (x2 – x1)
b = y – mx

Then solve the two equations and two unknowns to find the x, y coordinate of **Px**.

Given:

$\mathbf{y = m_1x + b_1}$
$\mathbf{y = m_2x + b_2}$

Substitute $\mathbf{m_1x + b_1}$ for **y**:
$\mathbf{m_1x + b_1 = m_2x + b_2}$
$\mathbf{m_1x – m_2x = b_2 – b_1}$
$\mathbf{x(m_1 – m_2) = b_2 – b_1}$
$\mathbf{x = (b_2 – b_1) / (m_1 – m_2)}$

Rewrite equation of line as $\mathbf{x = (y - b_1) / m_1}$
Substitute $\mathbf{(y - b_1) / m_1}$ for **x**:
$\mathbf{y = m_2((y - b_1) / m_1) + b_2}$
$\mathbf{y = ((m_2y - m_2b_1) / m_1) + b_2}$
$\mathbf{m_1y = m_2y - m_2b_1 + m_1b_2}$
$\mathbf{m_1y - m_2y = m_1b_2 - m_2b_1}$
$\mathbf{y(m_1 - m_2) = m_1b_2 - m_2b_1}$
$\mathbf{y = (m_1b_2 - m_2b_1) / (m_1 - m_2)}$

$\mathbf{Px = ((b_2 - b_1) / (m_1 - m_2), (m_1b_2 - m_2b_1) / (m_1 - m_2))}$

Now that the start and end points of the arc as well as the arcs apex have been calculated, the last remaining piece is to calculate the bounding box that is needed for the Windows GDI **Arc** function. The way it works is to imagine that the arc was actually tracing out a complete circle. Find the bounding box that precisely fits the imaginary circle then align that bounding box with the x and y coordinates **(See Figure 7.8)**.

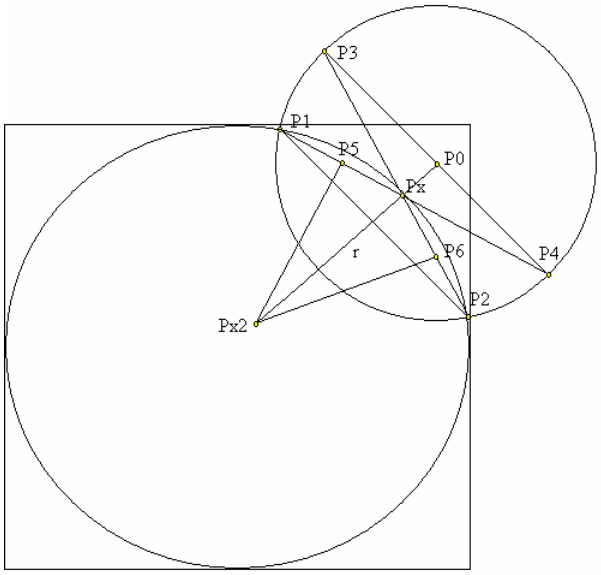


Figure 7.8. Circular Facet (Arc Bounding Rectangle)

Before the bounding box can be calculated a few more pieces of information are required. The center of the imaginary circle needs to be calculated as well as the circles radius. To calculate the center of the circle it is necessary to find the midpoint between lines **P1Px** and **P2Px** and label those points **P5** and **P6** respectively. **P5** and **P6** can be calculated using the midpoint formula.

MidPoint = ((X1 + X2) / 2, (Y1 + Y2) / 2)

Next calculate a line that is perpendicular to **P1Px** and passing through **P5** and calculate another line perpendicular to **P2Px** and passing through point **P6**. The intersection of these lines will be the center of the circle labeled **Px2 (See Figure 7.8)**. The slope of line **P1Px** is the same as line **P1P4** which was calculated previously and the slope of line **P2Px** is the same as line **P2P3** which was also calculated previously. Since we know the slope of these lines we can determine the slope of the perpendicular line by using the fact that the slope of the perpendicular line is the inverse reciprocal of the slope.

m1 = -(1/m2)

Now the y intercept can be calculated by using the slope that was just calculated and by plugging in the points P5 and P6 into the y intercept formula.

b = y – mx

Again we will have the equation of two lines with two unknowns so it can be solved as shown previously to acquire the intersection of the lines which will be the origin of the imaginary circle.

$$y = m_1x + b_1$$
$$y = m_2x + b_2$$
$$Px2 = ((b_2 - b_1) / (m_1 - m_2), (m_1b_2 - m_2b_1) / (m_1 - m_2))$$

The last piece of information required to find the bounding box of the imaginary circle is the radius of the circle. The radius can be calculated by using the distance formula between point **Px2** and **Px**.

d = Sqr((x2 – x1)^2 + (y2 – y1)^2)

Now it is a simple process to calculate the upper left and lower right coordinates of the bounding box. To find the x coordinate of the upper left corner subtract the radius from the x coordinate part of the center of the circle. To find the y coordinate of the upper left corner add the radius to the y coordinate part of the center of the circle. To calculate the lower right coordinate of the bounding box follow the same procedure of using the center of the circle and adding or subtracting the radius.

Finally, the arc can be drawn with the Windows GDI Arc function as follows:

```
// Draw arc in bounding rectangle with center at px2 and radius r
Arc(hdc,   // handle to device context
   (int)(px2.x - r),   // x-coord of rectangle's upper-left corner
   (int)(px2.y + r),   // y-coord of rectangle's upper-left corner
   (int)(px2.x + r),   // x-coord of rectangle's lower-right corner
   (int)(px2.y - r),   // y-coord of rectangle's lower-right corner
   (int)p1.x,   // x-coord of first radial ending point
   (int)p1.y,   // y-coord of first radial ending point
   (int)p2.x,   // x-coord of second radial ending point
   (int)p2.y);  // y-coord of second radial ending point
```

7.2.5 Drawing The Circular Graph

The circular graph is actually drawn onto the texture with a call to the ***Draw*** **(See Listing 7.5)** method of the ***CircularGraph*** class. When the ***Draw*** method is called the program calculates all of the arcs up to the current step position. That is to say if the sequence is 25 nucleotides long and the step position is currently at position 20 of 25 then the ***Draw*** method will draw the circular graph with all the arc necessary up to position 20. The first parameter of the ***Draw*** method is a handle to a device context on which to draw so the final picture will be drawn onto the device context and would appear as in Figure 7.9.

```
void CircularGraph::Draw(HDC hdc, char* Sequence, int iStep, int*
Pairing,  float xsize, float ysize)
```

Listing 7.5. CircularGraph.Draw Method

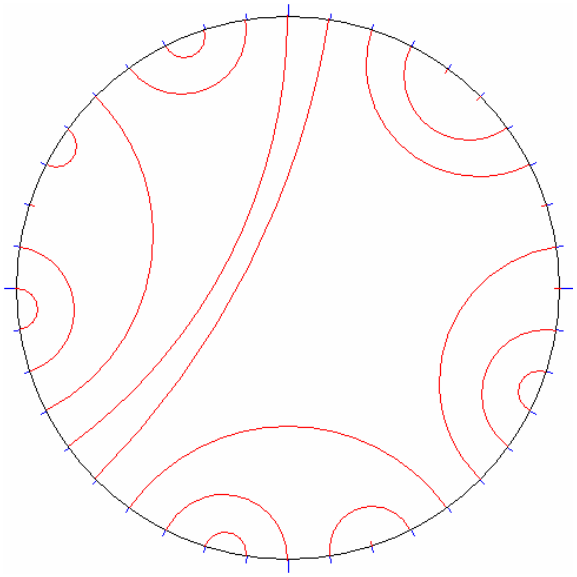


Figure 7.9. Circular Graph

7.3 Planar Graph Calculation

The planar graph calculation as described in section 3.5.4 is the closest approximation to how the secondary structure would actually appear under natural conditions. The planar graph usually consists of the letters A, C, G, and U which represent the nucleotide bases, positioned on a two dimensional graph in such a way as to represent their distance from each other as well as their pairing. Sometimes the bases are omitted and lines are draw to represent the secondary structure where a joint in the line represents the place where a nucleotide base would be positioned. The pairing between the bases is represented by a blue circle for an A-U bond and a red circle for a G-C bond. Sometimes the circles are replaced with red or blue lines. The planar graph representation has numbers which represent the loops that are formed in the sequence. For larger sequences a positional number is placed outside the loop at a given interval so that the linear position in the sequence can be easily determined.

7.3.1 Planar Graph Base Program

The base of the planar graph calculation used in the visualization was originally written by Robert E. Bruccoleri. Bruccoleri wrote a program called NAView which stands for nucleic acid view **[29]**. The NAView program takes a Zuker .CT file as input, runs through its calculation and outputs a .PLT2 plotter file. If another output type is needed then a companion program written by Darrin Stewart could be used to convert the plotter file to a postscript file. Darrin Stewart wrote a program called PLT22PS which takes the .PLT2 output from NAView as input and Darrin's program outputs a .PS postscript file. From this point the output could be printed or another program such as Adobe Acrobat Distiller could be used to convert the postscript file into a .PDF (portable document format) file which can be viewed with the Adobe Acrobat Reader **(See Figure 7.10)**.

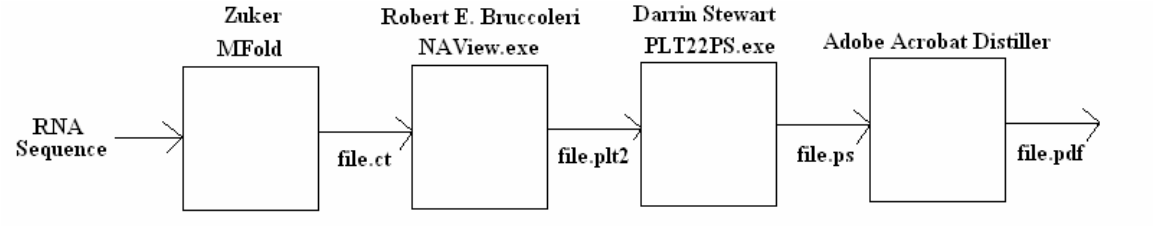


Figure 7.10. Process Required To Create PDF

The visualization program needed to be able to calculate the secondary structure of any sequence and output the secondary structure as a bitmap. Additionally, the visualization program needed to be extremely efficient so the use of an external program was not feasible. Based on these two requirements it was necessary to take elements from the NAView program and the PLT22PS program and incorporate them into the visualization. The NAView program was turned into a C++ class called RNAGraph which is the heart of the calculation. The main modification to the program was that instead of taking a Zuker .CT file as input it was modified so that it could also take input passed into it through an interface method. It was also modified so that instead of instantly sending its output to a plotter .PLT2 file it stored its output in memory. Next the RNAGraph class what extended through inheritance into a class called RNAGraphBMP. The extended class took some drawing elements from the PLT22PS program so that it was able to take the output from RNAGraph and calculate a bitmap in memory **(See Figure 7.11)**. The bitmap could then be used as part of the texture that gets drawn on the stage object.

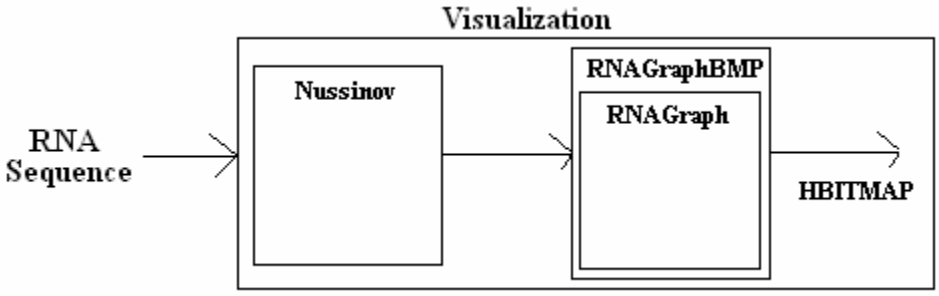


Figure 7.11. Sequence to Bitmap

7.3.2 Planar Graph Calculation Input

There are four elements which are needed as input to the planar graph calculation. First, an integer is needed which holds the length of the sequence. Second, a character array is needed which holds the title of the given sequence. Third, a character array is needed which holds the actual sequence. And fourth, an integer array is needed which holds the pairing in the sequence. This pairing array is the same as the input to the circular graph in section 7.2.1.

```
int   Length   = 9;
char* Title    = [‘Y’, ‘e’, ‘a’, ‘s’, ‘t’]
char* Sequence = [‘G’, ‘G’, ‘G’, ‘A’, ‘A’, ‘A’, ‘U’, ‘C’, ‘C’]
int*  Pairing  = [ 0,   8,   7,   3,   4,   6,   5,   2,   1]
```

7.3.3 Planar Graph Walkthrough

The planar graph calculation starts with a call to the *LoadFromRNASequence* method **(See Listing 7.6)**. This method is used as the gateway where the input to the calcualation gets passed into the RNAGraph class. The four variables being passed into the method are as descibed in section 7.3.2.

```
void RNAGraph::LoadFromRNASequence(char* Title, char* Sequence, int
Length, int* Pairing)
```

Listing 7.6. LoadFromRNASequence Method Prototype

The *LoadFromRNASequence* method starts out by dynamically creating an array of structures of type Base **(See Listing 7.7).** The array labeled as ***bases*** is dynamically create by calling the c++ malloc function. The ***bases*** array is one element longer than the length of the input sequence and that is so that the element at array index zero can be used to store a start symbol. Type start symbol will be represented on the planar graph by a small black dot and it is mearly a place holder that allows you to visually determine where the sequence starts.

```
typedef struct base_tag {
      char name;
      int mate, hist_num;
      float x, y;
      bool extracted;
      struct region_tag *region;
}Base;
```

Listing 7.7. Structure of type Base

The *LoadFromRNASequence* method then enters a loop where the member variables of each structure in the array are initialized. The ***name*** variable is used to store the nucleotide base symbol so each array element is updated with the sequence symbols. The ***mate*** variable is used to store the index of the nucleotide base which bonds with the current base. If the nucleotide base does not have a bonding pair then ***mate*** is populated with zero. The ***hist_num*** variable is simply the linear position of the structure in the array. The variables ***x*** and ***y*** are initialized with the fixed number 9999.0 but they are updated later in the calculation. These ***x*** and ***y*** variables are the main output of the calculation and will store the two dimensional position of the nucleotide base in the resulting graph. The two remaining variables, ***extracted*** and ***region*** are initialized to false and null respectively and are used later in the calculation. The resulting array created in this first method is as follows **(See Figure 7.12)**:

```
nbase = 9
bases =(struct Base *) malloc(sizeof(struct Base) * (nbase + 1));
```

	0	1	2	3	4	5	6	7	8	9
char name	'o'	'G'	'G'	'G'	'A'	'A'	'A'	'U'	'C'	'C'
int mate	0	0	9	8	0	0	7	6	3	2
int hist_num	0	2	3	4	5	6	7	8	9	10
float x	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999
float y	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999
bool extracted	False	False	False	False	False	False	False	False	False	False
region* region	Null	Null	Null	Null	Null	Null	Null	Null	Null	Null

Figure 7.12. Array of Base structures

Now that the ***bases*** array has been created and initialized the calculation goes into a method where it tries to find regions within the ***bases*** array. A region is section of the sequence that is connected in some way such as a stem or the start and ending pairs of a loop. The region finding method starts out by dynamically creating two arrays. The first array label as the ***mark*** array is an array of boolean varaibles and is only used for the region finding phase of the calculation and then it is removed. The ***mark*** array is used to

determine if the element at any given index in the array has been found to be part of a region. If the mark variable at the given index is true then the element is already part of the region and the calculation moves on to the next element. The second array label as the ***regions*** array in an array of Region structures **(See Listing 7.8)** and is used to store the start and end positions of the regions withing the sequence.

```
typedef struct region_tag {
      int start1, end1, start2, end2;
}Region;
```

Listing 7.8. Structure of type Region

The calculation then enters another loop where it goes through the ***bases*** array and updates the ***regions*** array with all regions that are found to exist. During this process the ***region*** pointer variable in the ***bases*** array also gets updated so that it points to the matching index in the ***regions*** array. As can be seen in Figure 7.4 there is a stem region that starts at ***bases*** array index 2 and ends at index 3 and the corresponding nucleotides start at ***bases*** array index 8 and ends at index 9. This is reflected in the ***regions*** array where a region has been determined to exist so the varaibles start1, end1, start2 and end2 are updated with those array indeces. A second region for the given sequence was found to exist at bases array index 6 and ending at index 7. Therefore, the arrays that exist after the region finding phase are as follows **(See Figure 7.13)**:

nbase = 9

bases

	0	1	2	3	4	5	6	7	8	9
char name	**'o'**	**'G'**	**'G'**	**'G'**	**'A'**	**'A'**	**'A'**	**'U'**	**'C'**	**'C'**
int mate	**0**	**0**	**9**	**8**	**0**	**0**	**7**	**6**	**3**	**2**
int hist_num	**0**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
float x	**9999**	**9999**	**9999**	**9999**	**9999**	**9999**	**9999**	**9999**	**9999**	**9999**
float y	**9999**	**9999**	**9999**	**9999**	**9999**	**9999**	**9999**	**9999**	**9999**	**9999**
bool extracted	**False**	**False**	**False**	**False**	**False**	**False**	**False**	**False**	**False**	**False**
region* region	**Null**	**Null**	**R[0]**	**R[0]**	**Null**	**Null**	**R[1]**	**R[1]**	**R[0]**	**R[0]**

```
mark = (bool *) malloc(sizeof(int) * (nbase + 1));
```

	0	1	2	3	4	5	6	7	8	9
bool mark	**False**	**False**	**True**	**True**	**False**	**False**	**True**	**True**	**True**	**True**

nregion = 2

```
regions = (struct Region *) malloc(sizeof(struct Region)*(nbase + 1));
```

	0	1	2	3	4	5	6	7	8	9
int start1	**2**	**6**								
int end1	**3**	**6**								
int start2	**8**	**7**								
int end2	**9**	**7**								

Figure 7.13. Arrays after region finding phase

At this point both the ***bases*** array and the ***regions*** array have been initialized and all regions have been identified. The next phase of the calculation is to determine the loops that exist in the given sequence. The calculation now enters another method where all the previous input is used to identify the loops. The process starts by dynamically creating an array of Loop **(See Listing 7.9)** structures labeled as ***loops*** which is again one more than the length of the sequence. A recursive method is then executed which goes through the ***bases*** and ***regions*** arrays to determine where loops exist. As the loops are determined the calculation dynamically creates another array of Connection **(See Listing 7.10)** structures which holds the connections between the loops.

```
typedef struct loop_tag {
      int nconnection;
      struct connection_tag **connections;
      int number;
      int depth;
      bool mark;
      float x, y, radius;
}Loop;
```

Listing 7.9. Structure of type Loop

```
typedef struct connection_tag {
      struct loop_tag *loop;
      struct region_tag *region;
      int start, end;    // Start and end form the 1st base pair of the region.
      float xrad, yrad, angle;
      bool extruded;     // True if segment between this connection and
                         // the next must be extruded out of the circle
      bool broken;       // True if the extruded segment must be drawn long.
}Connection;
```

Listing 7.10. Structure of type Connection

As the recursive method progresses, loops will be identified and stored into the ***loops*** array. The loops array has member variables that will store the ***x*** and ***y*** coordinate of the loop as well as the ***radius*** of the loop. The x,y coordinates and the radius are used later in the calculation to determine the position of the nucleotide bases around the loop of a given radius. The ***loops*** array also stores a pointer to a ***connection*** array which connects the loops to each other as well as the regions. The arrays that are setup after this phase of the calculation are as follows **(See Figure 7.14)**:

nbase = 9
bases

	0	1	2	3	4	5	6	7	8	9
char name	'o'	'G'	'G'	'G'	'A'	'A'	'A'	'U'	'C'	'C'
int mate	0	0	9	8	0	0	7	6	3	2
int hist_num	0	2	3	4	5	6	7	8	9	10
float x	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999
float y	9999	9999	9999	9999	9999	9999	9999	9999	9999	9999
bool extracted	False	False	False	False	False	False	False	False	False	False
region* region	Null	Null	R[0]	R[0]	Null	Null	R[1]	R[1]	R[0]	R[0]

nregion = 2
regions

	0	1	2	3	4	5	6	7	8	9
int start1	2	6								
int end1	3	6								
int start2	8	7								
int end2	9	7								

loop_count = 0
loops = `(struct Loop *) malloc(sizeof(struct Loop) * (nbase + 1));`

	0	1	2	3	4	5	6	7	8	9
int nconnection	1	2	1							
connection**	**A	**B	**C							
int number	1	2	3							
int depth	0	0	0							
bool mark	True	True	True							
float x	0.0	0.0	0.0							
float y	0.0	0.0	0.0							
float radius	0.0	0.0	0.0							

connection = `(struct Connection **)realloc(retloop->connections,`
`(++retloop->nconnection + 1) * sizeof(struct Connection *));`
A

	0	1
Loop* loop	Loop[2]	Null
Region* region	Region[0]	
int start	2	
int end	9	
float xrad	0.0	
float yrad	0.0	
float angle	0.0	
bool extruded	False	
bool broken	False	

connection = `(struct Connection **)realloc(retloop->connections,`
`(++retloop->nconnection + 1) * sizeof(struct Connection *));`
B

	0	1	2
Loop* loop	Loop[3]	Loop[1]	Null
Region* region	Region[1]	Region[1]	
int start	6	8	
int end	7	3	
float xrad	0.0	0.0	
float yrad	0.0	0.0	
float angle	0.0	0.0	

bool extruded	False	False	
bool broken	False	False	

```
connection = (struct Connection **)realloc(retloop->connections,
(++retloop->nconnection + 1) * sizeof(struct Connection *));
C
```

	0	1
Loop* loop	Loop[2]	Null
Region* region	Region[1]	
int start	7	
int end	6	
float xrad	0.0	
float yrad	0.0	
float angle	0.0	
bool extruded	False	
bool broken	False	

Figure 7.14. Arrays after loop finding phase

At this point the calculation now enters another recursive method called ***traverse_loop*** **(See Listing 7.11)**. This method is really the heart of the calculation. Everything previous to this method was identifying regions, loops and connections and preparing the necessary data required for this calculation. The ***traverse_loop*** method uses all the arrays and recursively travels through the arrays. As the method travels through the arrays it updates all the coordinate and radius information about where the objects should be positioned on the resulting planar graph. The method does not return anything and therefore its return type is void. What the methods output is, is that it updates all of the member variables in all the arrays. When this method completes all information necessary to draw the planar graph is present.

```
void RNAGraph::traverse_loop(struct Loop *lp, struct Connection
*anchor_connection)
```

Listing 7.11. traverse_loop method prototype

7.3.4 Drawing The Planar Graph

Since all information necessary to draw the planar graph has now been calculated the program now enters its last phase which is to actually draw the planar graph. The drawing functionality is actually located in a class called ***RNAGraphBMP*** which was inherited from the ***RNAGraph*** class. The ***RNAGraphBMP*** class has access to all the

arrays and variables which have been created in the previous calculations and uses that information to draw the planar graph.

To actually draw the planar graph on to the texture a call to the ***Draw*** **(See Listing 7.12)** method of the ***RNAGraphBMP*** calss is used. The ***Draw*** method takes 13 parameters on its method call. Most of the parameters are used in order to turn on or off certain features in the planar graph such as marking the loops or drawing a cirucle or line between bases.

```
void RNAGraphBMP::Draw(HDC hdc, float BMPScale, float xsize, float
ysize, bool mark_loops, bool draw_bases, float csz, int label_rate,
bool dot_pairs, int mosaicx, int mosaicy, float glob_rot, RNA *pRNA)
```

Listing 7.12. RNAGraphBMP.Draw Method

In order to draw the image in a desired height and width the ***Draw*** method inputes the desired image size through its ***xsize*** and ***ysize*** parameters. The ***RNAGraph*** class calculated all of the coordinates necessary to draw the planar graph but the calculation used its own internal units. That means that the ***Draw*** method has to setup vectors for scaling and translation in order to properly draw the image. The ***Draw*** method follows this general order of events. It first sets up the scaling and tranlation vectors. Next it draws the Sequence Title centered across the bottom of the image and scaled accordingly. Next it draw the small circle which represents the start of the sequence. Next it draws the sequence lines, the lines that connect the nucleotide bases. Next it draws the circles or lines between the pairs which have a bond. Next it draws the nucleotide base symbol. Next it draws the linear position number based on the label rate. And finally it draws the loop numbers. All of this is drawn onto the device context which was passed into the method and the result is as follows **(See Figure 7.15)**:

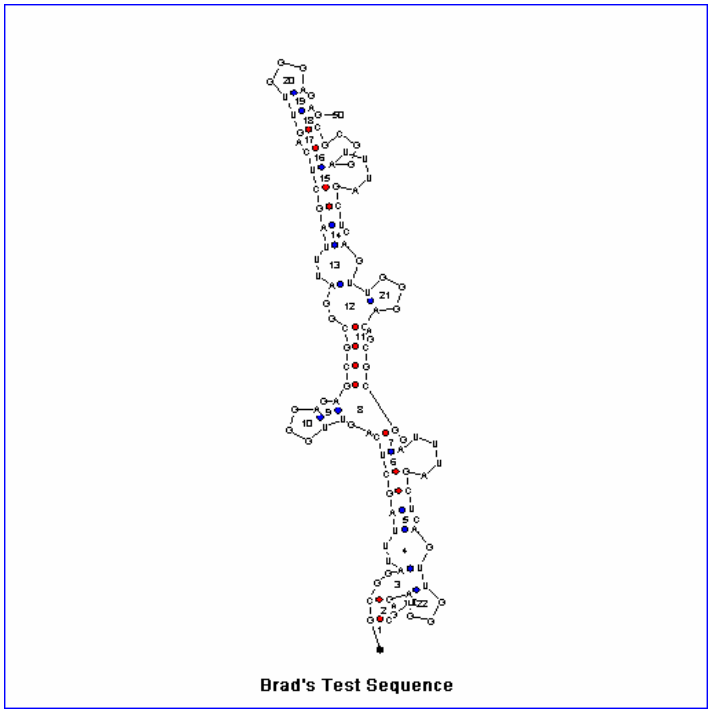


Figure 7.15. Planar Graph

7.4 Matrix Graph Calculation

The matrix graph is the graphical representation of the dynamic programming matrix created in the Nussinov secondary structure algorithm. The matrix graph consists of a two dimensional rectangular grid where each element in the grid contains a score from the Nussinov algorithm. The RNA sequence is aligned with the matrix across the top and along the left side of the matrix. The Nussinov algorithm uses an upper triangular matrix so that portion of the matrix will be the portion that has significant data. There are two versions of the matrix graph that are used in the visualization based on the version of the Nussinov algorithm selected. If the standard Nussinov algorithm is selected then each cell in the matrix will contain an integer value which represents the intermediate score of the Nussinov algorithm. If the SCFG Nussinov algorithm is selected then each cell of the matrix will contain a floating point number which is based on the probability calculation used in the algorithm. Finally, the matrix will have cells with colored backgrounds which correspond to the traceback path of the algorithm. These colored cells use a color scale to indicate the distance traveled along the traceback path.

7.4.1 Drawing The Matrix Graph

The matrix graph is drawn by calling the ***Draw*** **(See Listing 7.13)** method of the ***MatrixGraph*** class. The draw method takes four parameters which are used to supply all information necessary to draw the graph. The ****pRNA*** parameter supplies all information about the RNA sequence such as the nucleotides in the sequence the sequence length and the current step position. The ****pNussinov*** parameter supplies all information about which version of the Nussinov algorithm has been selected (Standard or SCFG) as well as the dynamic programming matrix. The ***xsize*** and ***ysize*** parameters are used to supply the height and width of the image that should be drawn. Finally, ***hdc*** is a handle to the device context on which the image will be drawn.

```
void MatrixGraph::Draw(HDC hdc, RNA *pRNA, Nussinov *pNussinov, float
xsize, float ysize)
```

Listing 7.13. MatrixGraph.Draw Method

The ***Draw*** method starts out by calculating the cell dimensions based on the image size variables ***xsize*** and ***ysize***, and dividing that by the length of the RNA sequence. That calculation will return the cell height and with needed so that the whole matrix will fit within the image size. Next, the ***Draw*** method goes into a loop which draws the RNA sequence across the top and left borders. The loop starts with the nucleotide base, draws that base scaled to a size that fits within the cell, then moves one cell distance down and starts again. Once the RNA sequence has been drawn then the program enters a loop nested in another loop. The outer loop travels down the columns of the matrix while the inner loop travels down the rows. As the program comes across a cell with a value from the dynamic programming matrix it takes that value, converts it to a string, scales it so that it fits within the cell and draws the value. If the current cell is determined to be on the traceback path then the background of the cell is colored according the current color scale position. The final output from the calculation is as follow **(See Figure 7.16)**:

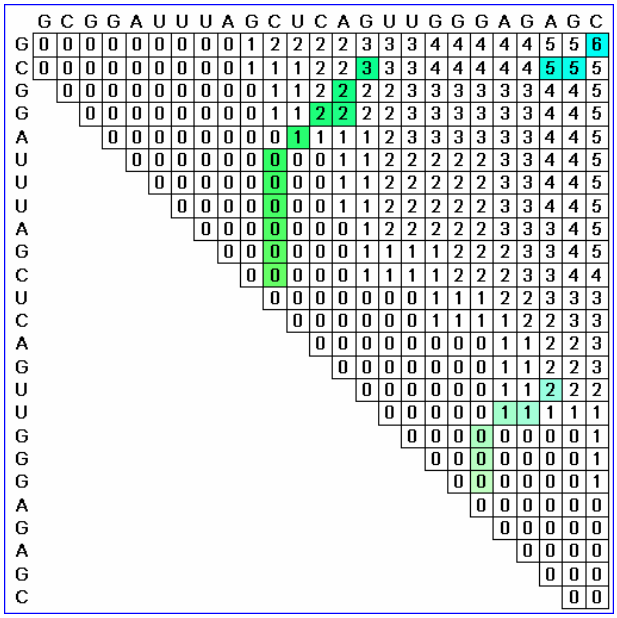


Figure 7.16. Matrix Graph

7.5 Bracketed Graph Calculation

The main purpose of the bracketed graph representation is to visually convey the bonds between the nucleotide bases in the RNA sequence. The bonds are displayed in the graph by aligning a bracket with the RNA sequence. An open bracket indicates the first nucleotide base of the pair and the close bracket indicates the second base in the pair. A dash in the sequence indicates that no bond exists at the current location.

Since the bracketed graph representation doesn't take a large area to represent there was room on this facet to display additional information. The facet also contains the traceback path which is the result of the Nussinov algorithm. In addition, the facet contains some statistics such as which Nussinov algorithm was used, Standard or SCFG, the length of the sequence, the current step position within the sequence and the optimal score of the Nussinov algorithm.

7.5.1 Drawing The Bracketed Graph

The bracketed graph is drawn through a call to the ***Draw*** **(See Listing 7.14)** method in the ***BracketedGraph*** class. The method takes five parameters and returns a void. The results of the ***Draw*** method are that the image is drawn onto the ***hdc*** parameter. The ***hdc*** parameter is a handle to a device context which is a bitmap that eventually becomes the texture. The other parameters ****pRNA***, ****pNussinove***, ***xsize*** and ***ysize*** are used to gather the information used for the statistics as well as the traceback and bracketed graph representation.

```
void BracketedGraph::Draw(HDC hdc, RNA *pRNA, Nussinov *pNussinov,
float xsize, float ysize)
```

Listing 7.14. BracketedGraph.Draw Method

The ***Draw*** method works by determining the size of the image through the ***xsize*** and ***ysize*** parameters. Based on the size of the image the method starts by drawing the statistics at the top of the image. The algorithm type is obtained through the ****pNussinov*** class pointer and the sequence length, step position and optimal score are all obtained through the ****pRNA*** class pointer. The ***Draw*** method then enters a loop which is used to draw the traceback path. The traceback path is stored as a linked list which is referenced through the ****pNussinov*** class. The program loops through the list and draws the row and column position of the cell within the dynamic programming matrix. Finally, the ***Draw*** method enters another loop which is used to draw the bracketed representation. The bracketed graph is stored as an array of integers in the ****pRNA*** class and is drawn by drawing the sequence on a row and then aligning either a open bracket, close bracket or dash on the next row. The final output from the bracketed graph calculation is as follows **(See Figure 7.17)**:

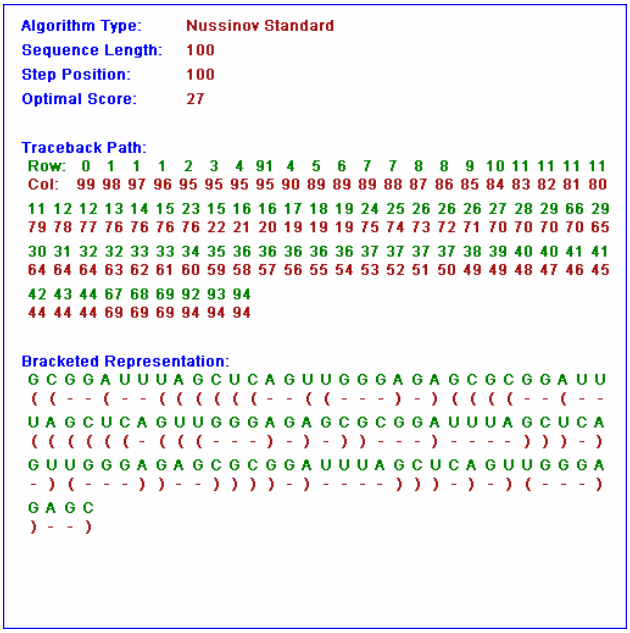


Figure 7.17. Bracketed Graph

7.6 Progress Bar Calculation

The progress bar representation is used to show the linear position of the current step within the RNA sequence. The progress bar works like a standard windows progress bar except that the progress is tied directly to a particular nucleotide base in the sequence. As the program steps through the sequence the current position is covered by the blue progress bar to indicate that the current nucleotide is part of the calculation. All nucleotides which are not covered with the blue bar are beyond the current step position and are not included in the Nussinov calculation.

7.6.1 Drawing The Progress Bar

The progress bar is drawn through the ***Draw*** **(See Listing 7.15)** method in the ***ProgressGraph*** class. The ***Draw*** method takes five parameters in order to supply the information necessary for the calculation. The result of the method is that the progress bar image is drawn onto the device context of a bitmap which is passed in as the ***hdc*** parameter. The ***Sequence***, ***RNALength*** and ***StepPostion*** are all parameters supplying information about the current state of the visualization.

```
void ProgressGraph::Draw(HDC hdc, char* Sequence, int RNALength, int
StepPosition, float xsize, float ysize)
```

Listing 7.15. BracketedGraph.Draw Method

The ***Draw*** method starts out by determining the cell size that each character in the sequence must fit into and then calculates how many cells per line will fit onto the progress bar based on the current ***xsize*** and ***ysize*** parameters. Once it is known how many cells fit per line then the ***Draw*** method enters a loop and starts to draw the RNA sequence. As each nucleotide base in the sequence is encountered the program determines if the nucleotide base is either within or beyond the current step position. If the nucleotide base is within the current step position then the base is drawn with a blue background so that it's part of the progress bar. If the base is beyond the step position then the base is drawn with a white background and is not part of the blue progress bar. When the loop hits the end of a line the program moves to the next line and starts again until the whole sequence has been covered. The output of the progress bar is as follows **(See Figure 7.18)**:

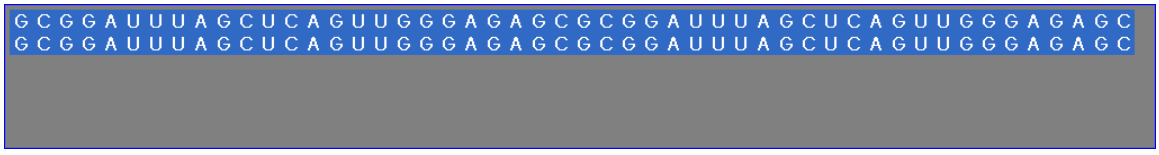


Figure 7.18. Progress Bar

Chapter 8

Program Usage

This chapter gives an overview of program usage and requirements. A minimum recommended hardware and software requirement is detailed. The chapter finishes by detailing the usage of the visualization program.

8.1 Program Requirements

The visualization program was written in C++ using the Microsoft Visual Studio 2005 development environment which is dependent on the Microsoft .Net 2.0 framework. This dependency means that the visualization requires that the .Net 2.0 framework has been installed. The visualization also uses Microsoft DirectX 9.0c in order to handle the three dimensional drawing functions so DirectX 9.0c must be installed. Both the .Net 2.0 framework and DirectX 9.0c are free downloads which can be obtained through the Microsoft website.

Since the visualization utilizes a three dimensional interface, the program has to run on fairly strong hardware. Although the program will run on a computer without a 3D accelerator card through software emulation, it is recommended that the computer have a 3D accelerator card. Therefore, recommended hardware would be a computer with a 2.0GHz CPU, 512MB RAM and a 3D accelerated graphics card.

With the release of Microsoft's next version of Windows code named Vista, the DirectX API will be built into the operating system [30]. This means that with very minimal program modification to reference the new DirectX 10 API, the operating system will automatically support the three dimensional components of the visualization.

8.2 Program Usage

In order to use the visualization, the program should first be launched by executing the RNAVis.exe program. Upon launching the visualization, the program will automatically load an example RNA sequence into the display interface and all the visualization options will be available **(See Figure 8.1)**. In the upper left corner of the interface there is a button that allows the help commands to be toggled on or off. The upper right corner of the interface contains the Nussinov type buttons as well as the algorithm stepping buttons. The main middle section of the interface contains the visualization of the Nussinov algorithm. The lower left corner of the interface displays statistics about the visualization. Finally, the lower right corner of the interface has three buttons for visualization options, a button to toggle between full screen, and a button for Direct3D options.

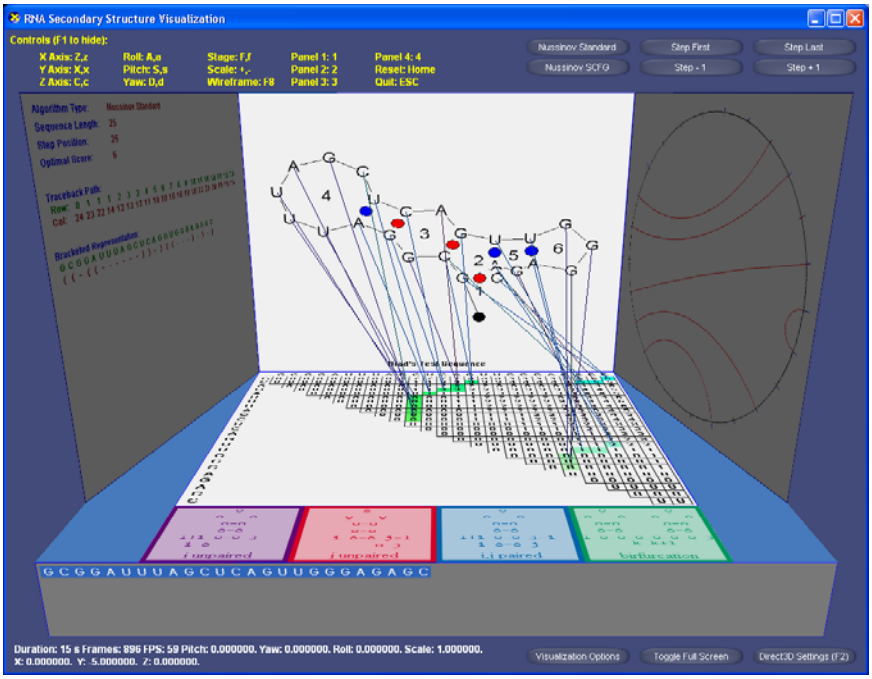


Figure 8.1. The main visualization interface

From this point one could step through the example sequence by clicking any one of the four stepping buttons or a new sequence could be loaded into the visualization. To load a new sequence one of two buttons could be clicked depending upon whether the standard or SCFG versions of the Nussinov algorithm are desired. After clicking the desired Nussinov type button, a new window will pop up which has several options related to the type of Nussinov algorithm selected.

If the standard Nussinov algorithm was selected then a window is displayed that allows one to enter the RNA sequence as well as several parameters related to the standard algorithm **(See Figure 8.2)**. To give the sequence a name one would simply type in the sequence name into the edit box. To enter the RNA sequence one could either type the sequence directly into the edit box or one could paste a FASTA sequence into the box. To enter the scoring matrix one could enter any integer parameters into the scoring matrix fields but the visualization automatically defaults a scoring matrix upon program startup. In addition, there are some scoring matrix buttons which will populate the fields with commonly used parameters. Finally, there's an edit box that is used to enter the minimum length that a hairpin loop is allowed to be. The hairpin loop value should be an integer between 0 and 10.

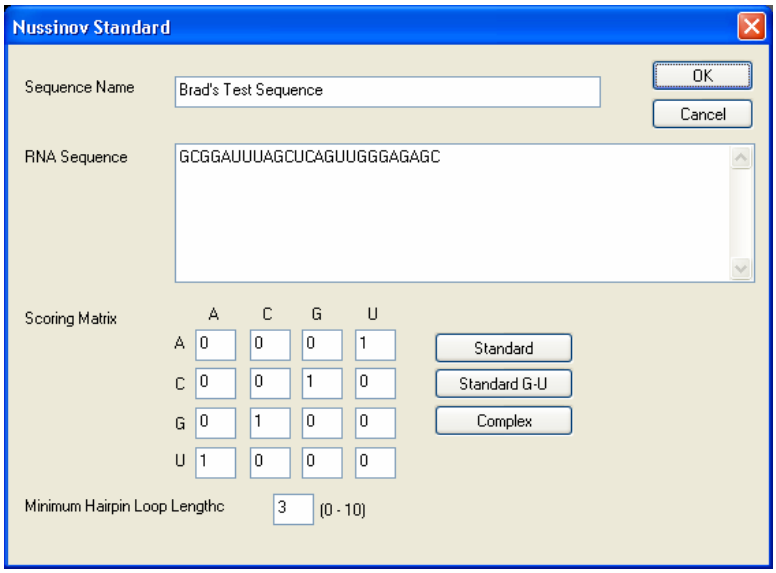


Figure 8.2. Standard Nussinov Options

If the SCFG Nussinov algorithm was selected then a window is displayed that allows one to enter the RNA sequence as well as several parameters related to the SCFG algorithm **(See Figure 8.3)**. Just as in the standard version, the SCFG version has edit boxes to enter the sequence name as well as the sequence its self. The SCFG pop up has additional edit boxes to enter all of the probabilities for the grammar options. All together there are thirteen edit box related to the probabilities any of which can be modified from the default which is populated at program startup. Finally, there is an edit box for the minimum hairpin loop length just like the one available in the standard version.

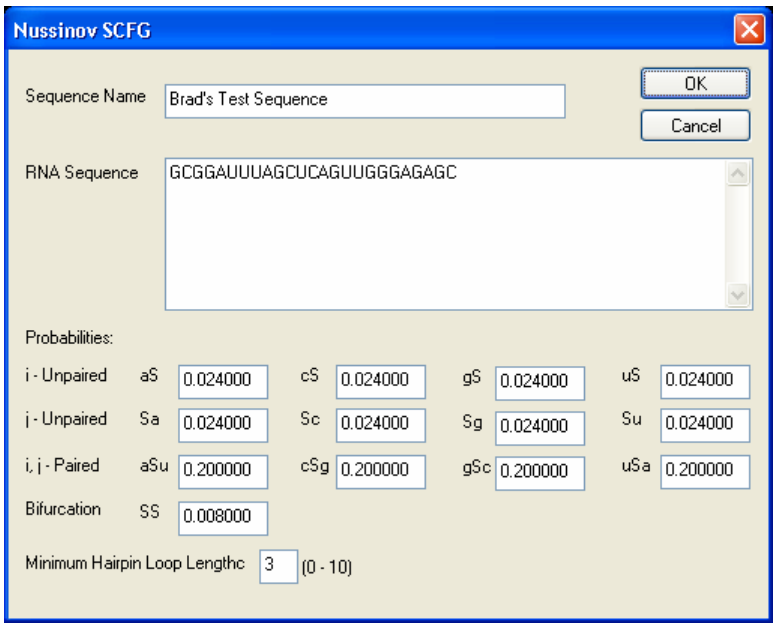


Figure 8.3. SCFG Nussinov Options

At the bottom left of the main visualization interface there is a button for visualization options. When this button is clicked a pop up windows is displayed that contains several options which can be set **(See Figure 8.4)**. There is a check box which is used to toggle the three dimensional lines on or off between the matrix graph and the planar graph. The other four options in the window are all related to the planar graph. The ***Mark Loops*** check box toggles the drawing of an integer which indicates the loop number, on or off. The ***Draw Bases*** check box toggles the drawing of the nucleotide base symbol, on or off. The ***Dot Pairs*** check box toggles between drawing a circular dot or a line between the

nucleotide bases with a bond. Finally, the ***Label Rate*** edit box is used to enter the rate a which a linear position number will be drawn on the planar graph.

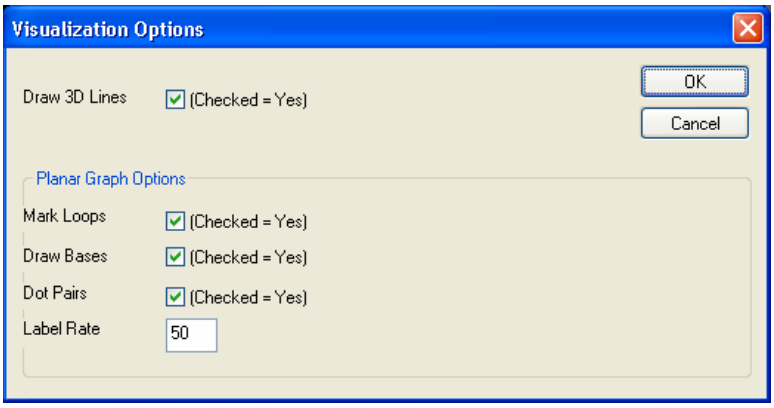


Figure 8.4. Visualization Options Window

The main interface also contains a button which is used for Direct3D Settings **(See Figure 8.5)**. This window was taken directly from an example project provided by Microsoft which was part of the DirectX SDK. This window has options for selecting the display adapter if more than one is present. The render device can be selected which can be either the hardware video card or software drivers. The Direct3D Settings window provides additional options all of which are related to Direct3D.

Figure 8.5. Direct3D Settings

Chapter 9

Conclusion

The objective of this project was to create a visualization that would be used as a teaching tool in order to increase the intuitive understanding of the Nussinov secondary structure prediction algorithm. In order to facilitate this objective the visualization took advantage of several sophisticated techniques with the purpose of conveying information about the algorithm. One of the primary techniques used to communicate the information about the algorithm was to illustrate the data with several representations simultaneously. The reasoning behind showing several representations simultaneously is to allow the user to draw connections between the data by visually analyzing the representations. The need to show several representations of data simultaneously led to the first innovation of the visualization. The program uses a three dimensional interface with the intention that the data representations can all be viewed together without cluttering the interface. A significant amount of attention was paid to creating an aesthetically pleasing interface which draws the attention of the user to the significant information while other ancillary information is conveyed on the perimeter. Additionally, the data representations where spatially aligned in such a way as to allow additional connections to be drawn by the visualization. For instance, the visualization draws three dimensional lines between the planar graph and the dynamic programming matrix in order to show how the calculation obtained the results. Another innovative feature of the visualization is its ability to allow the user to step through the RNA sequence and view intermediate results of the calculation. By starting with a small sequence and gradually stepping through the

sequence, information can be gleamed about the inner workings of the algorithm. Through the use of the techniques outlined here, the objective of creating a visualization teaching tool has been achieved.

Bibliography

[1] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. Biological Sequence Analysis, Probabilistic models of proteins and nucleic acids, Cambridge University Press. 1998.

[2] Setubal, J., and Meidanis, J., Introduction to Computational Molecular Biology, Brooks/Cole Publishing Company, 1997

[3] Pevsner, J., Bioinformatics and Functional Genomics, John Wiley & Sons, Inc., 2003

[4] Card, S., Mackinlay, J., and Shneiderman, B., Readings In Information Visualization, Using Vision To Think, Morgan Kaufmann Publishers, Inc., 1999

[5] Thorn, A., DirectX 9 Graphics, The Definitive Guide to Direct3D, Wordware Publishing, Inc., 2005

[6] De Stefani, A., RNA Visualization, Institute of Computer Graphics, Vienna University of Technology, Vienna, Austria

[7] Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/RNA

[8] Nussinov, R., Piecznik, G., Grigg, J.R., and Kleitman, D.J., Algorithms For Loop Matching, SIAM Journal on Applied Mathematics, 1978

[9] Mills, G. C., Kenyon, D. The RNA World: A Critique, Review Article, Origins & Design 17:1, Access Research Network, Gordon C. Mills and Dean Kenyon, 1996

[10] Woese, C., The Genetic Code - The Molecular Basis For Genetic Expression, Harper & Row, New York, 1967

[11] RNA Types. http://users.rcn.com/jkimball.ma.ultranet/BiologyPages/T/Transcription.html#rRNA, 2006

[12] Wikipedia, Non-Coding RNA. http://en.wikipedia.org/wiki/Non-coding_RNA

[13] Gruner, W., Giegerich, R., Strothmann, D., Reidys, C., Weber, J., Hofacker, I., Stadler, P., and Schuster, P. Analysis of RNA Sequence Structure Maps by Exhaustive Enumeration. Santa Fe Institute Preprint 95-10-099

[14] Zuker, M. RNA Secondary Structure – Lectures by Dr. Michael Zuker, Bio-5495, Department of Mathematical Sciences Rensselaer Polytechnic Institute, Troy, NY, 200

[15] Westhof, E., and Auffinger, P. RNA Tertiary Structure, Encylopedia of Analytical Chemistry, John Wiley & Sons Ltd, Chichester, 2000

[16] Martinez, M. An RNA folding rule. Department of Biochemisty and Biophysics, University of California, San Francisco. IRL Pres Limited, Oxford, England 1983

[17] Schmitz, M., Steger, G. Description of RNA Folding by "Simulated Annealing", Journal of Molecular Biology, Volume 255, Issue 1, 12 January 1996, Elsevier B.V., 2006

[18] Fekete, M. Prediction of RNA Secondary Structures Using Parallel Computers, 1997

[19] University of Southern California, XRNA, http://rna.ucsc.edu/rnacenter/xrna/xrna.html (Free Software)

[20] Hochsmann, M., Toller, T., Giegerich, R., and Kurtz, S. Local Similarity in RNA Secondary Structures, University of Bielefeld, Germany, IEEE Computer Society, 2003

[21] RNA Structure Prediction, http://ludwig-sun2.unil.ch/~bsondere/nussinov/form.html#nussinov

[22] Hogeweg, P., and Hesper, B. Energy Directed Folding of RNA Sequences. Volume 12 Number 1, IRL Press Limited, Oxford, England, 1984

[23] The GDI Coordinate System, http://www.functionx.com/visualc/gdi/gdicoord.htm (4/1/2006)

[24] Microsoft Visual Studio 2005 Documentation, Microsoft Corporation, 2006

[25] Kaiser, G.E., Doc Kaiser's, Microbiology Home Page, http://www.cat.cc.md.us/courses/bio141/lecguide/unit4/genetics/DNA/RNA/rna.html, 2005

[26] DNA, http://www.pembinatrails.ca/vincentmassey/topchem/DNA.html#nucleotides, 2006

[27] Microsoft DirectX 9.0C Documentation, Microsoft Corporation, 2006

[28] Andy Pike DirectX Tutorial, www.anypike.com/tutorials/directx8/006.asp , May 2006

[29] Robert E. Bruccoleri, NAView program, http://iubio.bio.indiana.edu:7780/archive/00000377/, May 2002

[30] Shah, Sarju and Yu, James, Windows Vista and DirectX 10, http://news.com.com/An+inside+look+at+Windows+Vista+-+page+4/2100-1043_3-6051736-4.html, March 21, 2006

[31] Vienna RNA Package, Representations Secondary Structures, http://www.tbi.univie.ac.at/~ivo/RNA/RNAlib/notations.html, 2006

[32] Angrish, Rohan, RNA Secondary Structure, CS262 – Computational Genomics, Lecture 17, May 2003.

```
1  //--------------------------------------------------------------------------------
2  // Copyright (c): 2006, All Rights Reserved
3  // Project:       SJSU Masters Project
4  // File:          RNAVis.cpp
5  // Purpose:       This is the main interface for the RNA Visualization program.
6  //                This program creates a 3D interface where several representations
7  //                of the RNA secondary structure can be viewed simultaneously.
8  //
9  // Start Date:    10/11/2006
10 // Programmer:    Brandon Hunter
11 //
12 // Based on the Basic starting point for new Direct3D samples
13 //
14 // Copyright (c) Microsoft Corporation. All rights reserved.
15 //--------------------------------------------------------------------------------
16
17 #include "dxstdafx.h"
18 #include "resource.h"
19 #include "Stage.h"              // Header file for stage class
20 #include "RNA.h"                // Header file for RNA class
21 #include "RNAGraphBMP.h"        // Header file for RNAGraphBMP class
22 #include "Nussinov.h"           // Header file for Nussinov class
23 #include "CircularGraph.h"      // Header file for CircularGraph class
24 #include "MatrixGraph.h"        // Header file for MatrixGraph class
25 #include "ProgressGraph.h"      // Header file for ProgressGraph class
26 #include "BracketedGraph.h"     // Header file for BracketedGraph class
27 #include "stdio.h"              // for the sprintf function used to convert float to char*
28 #include "LineList.h"           // Header for linked list of lines
29 #include <direct.h>             // Used for getcwd() function
30
31
32 //#define DEBUG_VS   // Uncomment this line to debug vertex shaders
33 //#define DEBUG_PS   // Uncomment this line to debug pixel shaders
34
35
36 //--------------------------------------------------------------------------------
37 // Global variables
38 //--------------------------------------------------------------------------------
39 ID3DXFont*              g_pFont = NULL;         // Font for drawing text
40 ID3DXSprite*            g_pTextSprite = NULL;   // Sprite for batching draw text calls
41 //ID3DXEffect*            g_pEffect = NULL;       // D3DX effect interface
42 //CModelViewerCamera      g_Camera;               // A model viewing camera
43 bool                    g_bShowHelp = false;    // If true, it renders the UI control text
44 CDXUTDialogResourceManager g_DialogResourceManager; // manager for shared resources of dialogs
45 CD3DSettingsDlg         g_SettingsDlg;          // Device settings dialog
46 CDXUTDialog             g_HUD;                  // dialog for standard controls
47 CDXUTDialog             g_SampleUI;             // dialog for sample specific controls
48
49 HBITMAP                 g_hbm = NULL;           // Handle to bitmap
50 Stage*                  g_pStage;               // The Stage object which acts as the background to draw on
51 RNA*                    g_pRNA;                 // Global pointer to RNA class
52 RNAGraphBMP*            g_pRNAGraphBMP;         // Global pointer to RNAGraphBMP class
53 Nussinov*               g_pNussinov;            // Global pointer to Nussinov class
54 CircularGraph*          g_pCircular;            // Global pointer to CircularGraph class
55 MatrixGraph*            g_pMatrix;              // Global pointer to MatrixGraph
```

```
    class
56 ProgressGraph*          g_pProgress;            // Global pointer to ProgressGraph   ↙
    class
57 BracketedGraph*         g_pBracketed;           // Global pointer to BracktedGraph   ↙
    class
58
59 DWORD m_dwFrames;
60 DWORD m_dwStartTime;
61 DWORD m_dwEndTime;
62 DWORD m_dwTotalPolygons;
63
64 // Globals For Visualization Options
65 bool    g_DrawLines = true; // Draw 3D lines from Matrix Graph to Planar Graph
66 bool    g_MarkLoops = true; // Draw loop numbers
67 bool    g_DrawBases = true; // Draw the base character
68 bool    g_DotPairs = true;  // Show bonded bases with a dot
69 int     g_LabelRate = 50;   // Rate at which to draw base number
70
71 //--------------------------------------------------------------------------------↙
    ---
72 // UI control IDs
73 //--------------------------------------------------------------------------------↙
    ---
74 #define IDC_TOGGLEFULLSCREEN     1
75 #define IDC_CHANGEDEVICE         2
76 #define IDC_NUSSINOVSTANDARD     3
77 #define IDC_NUSSINOVSCFG         4
78 #define IDC_STEPFIRST            5
79 #define IDC_STEPPREV             6
80 #define IDC_STEPNEXT             7
81 #define IDC_STEPLAST             8
82 #define IDC_VISUALIZATIONOPTIONS 9
83
84 //--------------------------------------------------------------------------------↙
    ---
85 // Forward declarations
86 //--------------------------------------------------------------------------------↙
    ---
87 bool    CALLBACK IsDeviceAcceptable(D3DCAPS9* pCaps, D3DFORMAT AdapterFormat,       ↙
    D3DFORMAT BackBufferFormat, bool bWindowed, void* pUserContext);
88 bool    CALLBACK ModifyDeviceSettings(DXUTDeviceSettings* pDeviceSettings, const    ↙
    D3DCAPS9* pCaps, void* pUserContext);
89 HRESULT CALLBACK OnCreateDevice(IDirect3DDevice9* pd3dDevice, const D3DSURFACE_DESC* ↙
    pBackBufferSurfaceDesc, void* pUserContext);
90 HRESULT CALLBACK OnResetDevice(IDirect3DDevice9* pd3dDevice, const D3DSURFACE_DESC*  ↙
    pBackBufferSurfaceDesc, void* pUserContext);
91 void    CALLBACK OnFrameMove(IDirect3DDevice9* pd3dDevice, double fTime, float       ↙
    fElapsedTime, void* pUserContext);
92 void    CALLBACK OnFrameRender(IDirect3DDevice9* pd3dDevice, double fTime, float     ↙
    fElapsedTime, void* pUserContext);
93 LRESULT CALLBACK MsgProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam, bool*   ↙
    pbNoFurtherProcessing, void* pUserContext);
94 void    CALLBACK KeyboardProc(UINT nChar, bool bKeyDown, bool bAltDown, void*       ↙
    pUserContext);
95 void    CALLBACK OnGUIEvent(UINT nEvent, int nControlID, CDXUTControl* pControl, void↙
    * pUserContext);
96 void    CALLBACK OnLostDevice(void* pUserContext);
97 void    CALLBACK OnDestroyDevice(void* pUserContext);
98 INT_PTR CALLBACK VisualizationOptions(HWND, UINT, WPARAM, LPARAM);
99 INT_PTR CALLBACK NussinovStandard(HWND, UINT, WPARAM, LPARAM);
100 INT_PTR CALLBACK NussinovSCFG(HWND, UINT, WPARAM, LPARAM);
101 int     checkFASTA(char* str);
102 int     checkRNA(char* str);
103 void    InitApp();
104 void    SetupCamera(IDirect3DDevice9* pd3dDevice);
105 void    Render3D(IDirect3DDevice9* pd3dDevice);
106 void    RenderText();
```

```
107 void    CreateDynamicTexture();
108 void    UpdateStageLines();
109 void    GetCurrentPath(char* buffer);
110
111 //--------------------------------------------------------------------------------
    ---
112 // Entry point to the program. Initializes everything and goes into a message
    processing
113 // loop. Idle time is used to render the scene.
114 //--------------------------------------------------------------------------------
    ---
115 INT WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int )
116 {
117     // Enable run-time memory check for debug builds.
118 #if defined(DEBUG) | defined(_DEBUG)
119     _CrtSetDbgFlag( _CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF );
120 #endif
121
122     // Set the callback functions. These functions allow DXUT to notify
123     // the application about device changes, user input, and windows messages.  The
124     // callbacks are optional so you need only set callbacks for events you're
    interested
125     // in. However, if you don't handle the device reset/lost callbacks then the
    sample
126     // framework won't be able to reset your device since the application must first
127     // release all device resources before resetting.  Likewise, if you don't handle
    the
128     // device created/destroyed callbacks then DXUT won't be able to
129     // recreate your device resources.
130     DXUTSetCallbackDeviceCreated(OnCreateDevice);
131     DXUTSetCallbackDeviceReset(OnResetDevice);
132     DXUTSetCallbackDeviceLost(OnLostDevice);
133     DXUTSetCallbackDeviceDestroyed(OnDestroyDevice);
134     DXUTSetCallbackMsgProc(MsgProc);
135     DXUTSetCallbackKeyboard(KeyboardProc);
136     DXUTSetCallbackFrameRender(OnFrameRender);
137     DXUTSetCallbackFrameMove(OnFrameMove);
138
139     // Show the cursor and clip it when in full screen
140     DXUTSetCursorSettings(true, true);
141
142     InitApp();
143
144     // Initialize DXUT and create the desired Win32 window and Direct3D
145     // device for the application. Calling each of these functions is optional, but
    they
146     // allow you to set several options which control the behavior of the framework.
147     DXUTInit(true, true, true); // Parse the command line, handle the default hotkeys
    , and show msgboxes
148     DXUTCreateWindow(L"RNA Secondary Structure Visualization");
149     DXUTCreateDevice(D3DADAPTER_DEFAULT, true, 1024, 768, IsDeviceAcceptable,
    ModifyDeviceSettings);
150
151     // Pass control to DXUT for handling the message pump and
152     // dispatching render calls. DXUT will call your FrameMove
153     // and FrameRender callback when there is idle time between handling window
    messages.
154     DXUTMainLoop();
155
156     // Perform any application-level cleanup here. Direct3D device resources are
    released within the
157     // appropriate callback functions and therefore don't require any cleanup code
    here.
158
159
160     if(g_pRNA != NULL)
161         delete g_pRNA;
```

```
162     if(g_pNussinov != NULL)
163         delete g_pNussinov;
164     if(g_pRNAGraphBMP != NULL)
165         delete g_pRNAGraphBMP;
166     if(g_pCircular != NULL)
167         delete g_pCircular;
168     if(g_pMatrix != NULL)
169         delete g_pMatrix;
170     if(g_pProgress != NULL)
171         delete g_pProgress;
172     if(g_pBracketed != NULL)
173         delete g_pBracketed;
174 
175     return DXUTGetExitCode();
176 }
177 
178 
179 //--------------------------------------------------------------------------------
    ---
180 // Initialize the app
181 //--------------------------------------------------------------------------------
    ---
182 void InitApp()
183 {
184     // Initialize dialogs
185     g_SettingsDlg.Init(&g_DialogResourceManager);
186     g_HUD.Init(&g_DialogResourceManager);
187     g_SampleUI.Init(&g_DialogResourceManager);
188 
189     g_HUD.SetCallback(OnGUIEvent);
190     g_HUD.AddButton(IDC_VISUALIZATIONOPTIONS, L"Visualization Options", 0, 0, 125,
    22);
191     g_HUD.AddButton(IDC_TOGGLEFULLSCREEN, L"Toggle Full Screen", 135, 0, 125, 22);
192     g_HUD.AddButton(IDC_CHANGEDEVICE, L"Direct3D Settings (F2)", 270, 0, 125, 22,
    VK_F2);
193 
194     g_SampleUI.SetCallback(OnGUIEvent);
195     int iX = 0, iY = 10;
196     g_SampleUI.AddButton(IDC_NUSSINOVSTANDARD, L"Nussinov Standard", iX, iY, 125, 22)
    ;
197     g_SampleUI.AddButton(IDC_NUSSINOVSCFG, L"Nussinov SCFG", iX, iY += 24, 125, 22);
198 
199     g_SampleUI.AddButton(IDC_STEPFIRST, L"Step First", iX += 135, iY = 10, 125, 22);
200     g_SampleUI.AddButton(IDC_STEPPREV, L"Step - 1", iX, iY += 24, 125, 22);
201 
202     g_SampleUI.AddButton(IDC_STEPLAST, L"Step Last", iX += 135, iY = 10, 125, 22);
203     g_SampleUI.AddButton(IDC_STEPNEXT, L"Step + 1", iX, iY += 24, 125, 22);
204 
205     m_dwFrames = 0;
206     m_dwStartTime = 0;
207     m_dwEndTime = 0;
208     m_dwTotalPolygons = 0;
209 
210     //Visualization started, so record time
211     m_dwStartTime = timeGetTime();
212 
213     char CurrentPath[150];
214     GetCurrentPath(CurrentPath); // use the function to get the path
215 
216     // Load the default image ("C:\\Thesis\\RNAVis\\StageTexture.bmp")
217     g_hbm = (HBITMAP) LoadImageA(0, strcat(CurrentPath, "\\StageTexture.bmp"),
    IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE);
218 
219     char* Title = new char[31]; //char* Title = (char *) malloc(sizeof(char) * 21);
220     strcpy(Title, "Default Visualization Sequence");
221     int Length = 25;
222     char* Sequence = new char[Length + 1]; //char* Sequence = (char *) malloc(sizeof
```

```
    (char) * Length);
223     strcpy(Sequence, "GCGGAUUUAGCUCAGUUGGGAGAGC");
224
225     g_pRNA = new RNA(Title, Sequence, Length); // Instantiate the RNA class object
226
227     g_pNussinov = new Nussinov(Nussinov::NussinovType::NussinovStandard, g_pRNA->
        getSequence(), g_pRNA->getLength());  // Instantiate an Nussinov class object
228     g_pNussinov->FillStage();
229     g_pNussinov->TraceBack(g_pRNA);  // Set the pairing array
230
231     g_pRNAGraphBMP = new RNAGraphBMP(0.7f);  // Instantiate the RNAGraphBMP class
        object
232     g_pRNAGraphBMP->LoadFromRNASequence(g_pRNA->getTitle(), g_pRNA->getSequence(),
        g_pRNA->getStepPosition(), g_pRNA->getPairing());
233
234     g_pCircular = new CircularGraph;  // Instantiate the CircularGraph class object
235     g_pMatrix = new MatrixGraph;  // Instantiate the MatrixGraph class object
236     g_pProgress = new ProgressGraph; // Instantiate the ProgressGraph class object
237     g_pBracketed = new BracketedGraph; // Instantiate the BracketedGraph class object
238
239     CreateDynamicTexture();
240 }
241
242
243 //--------------------------------------------------------------------------------
        ---
244 // Called during device initialization, this code checks the device for some
245 // minimum set of capabilities, and rejects those that don't pass by returning false.
246 //--------------------------------------------------------------------------------
        ---
247 bool CALLBACK IsDeviceAcceptable(D3DCAPS9* pCaps, D3DFORMAT AdapterFormat,
248                                  D3DFORMAT BackBufferFormat, bool bWindowed, void*
        pUserContext)
249 {
250     // Skip backbuffer formats that don't support alpha blending
251     IDirect3D9* pD3D = DXUTGetD3DObject();
252     if(FAILED(pD3D->CheckDeviceFormat( pCaps->AdapterOrdinal, pCaps->DeviceType,
253                     AdapterFormat, D3DUSAGE_QUERY_POSTPIXELSHADER_BLENDING,
254                     D3DRTYPE_TEXTURE, BackBufferFormat)))
255         return false;
256
257     return true;
258 }
259
260
261 //--------------------------------------------------------------------------------
        ---
262 // This callback function is called immediately before a device is created to allow
        the
263 // application to modify the device settings. The supplied pDeviceSettings parameter
264 // contains the settings that the framework has selected for the new device, and the
265 // application can make any desired changes directly to this structure.  Note however
        that
266 // DXUT will not correct invalid device settings so care must be taken
267 // to return valid device settings, otherwise IDirect3D9::CreateDevice() will fail.
268 //--------------------------------------------------------------------------------
        ---
269 bool CALLBACK ModifyDeviceSettings(DXUTDeviceSettings* pDeviceSettings, const
        D3DCAPS9* pCaps, void* pUserContext)
270 {
271     // If device doesn't support HW T&L or doesn't support 1.1 vertex shaders in HW
272     // then switch to SWVP.
273     if((pCaps->DevCaps & D3DDEVCAPS_HWTRANSFORMANDLIGHT) == 0 ||
274          pCaps->VertexShaderVersion < D3DVS_VERSION(1,1))
275     {
276         pDeviceSettings->BehaviorFlags = D3DCREATE_SOFTWARE_VERTEXPROCESSING;
277     }
```

```
278
279     // Debugging vertex shaders requires either REF or software vertex processing
280     // and debugging pixel shaders requires REF.
281 #ifdef DEBUG_VS
282     if( pDeviceSettings->DeviceType != D3DDEVTYPE_REF )
283     {
284         pDeviceSettings->BehaviorFlags &= ~D3DCREATE_HARDWARE_VERTEXPROCESSING;
285         pDeviceSettings->BehaviorFlags &= ~D3DCREATE_PUREDEVICE;
286         pDeviceSettings->BehaviorFlags |= D3DCREATE_SOFTWARE_VERTEXPROCESSING;
287     }
288 #endif
289 #ifdef DEBUG_PS
290     pDeviceSettings->DeviceType = D3DDEVTYPE_REF;
291 #endif
292
293     // For the first device created if its a REF device, optionally display a warning
        dialog box
294     static bool s_bFirstTime = true;
295     if(s_bFirstTime)
296     {
297         s_bFirstTime = false;
298         if(pDeviceSettings->DeviceType == D3DDEVTYPE_REF)
299             DXUTDisplaySwitchingToREFWarning();
300     }
301
302     return true;
303 }
304
305
306 //--------------------------------------------------------------------------------
        ---
307 // This callback function will be called immediately after the Direct3D device has
        been
308 // created, which will happen during application initialization and windowed/full
        screen
309 // toggles. This is the best location to create D3DPOOL_MANAGED resources since these
310 // resources need to be reloaded whenever the device is destroyed. Resources created
311 // here should be released in the OnDestroyDevice callback.
312 //--------------------------------------------------------------------------------
        ---
313 HRESULT CALLBACK OnCreateDevice(IDirect3DDevice9* pd3dDevice, const D3DSURFACE_DESC*
        pBackBufferSurfaceDesc, void* pUserContext)
314 {
315     HRESULT hr;
316
317     V_RETURN(g_DialogResourceManager.OnCreateDevice(pd3dDevice));
318     V_RETURN(g_SettingsDlg.OnCreateDevice(pd3dDevice));
319
320     // Initialize the font
321     V_RETURN(D3DXCreateFont(pd3dDevice, 15, 0, FW_BOLD, 1, FALSE, DEFAULT_CHARSET,
322                             OUT_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH |
        FF_DONTCARE,
323                             L"Arial", &g_pFont));
324
325     g_pStage = new Stage(pd3dDevice, 10.0f, 0.0f, -5.0f, 0.0f);
326     g_pStage->SetTextureFromBitmap(g_hbm);
327     UpdateStageLines();
328
329     // Define DEBUG_VS and/or DEBUG_PS to debug vertex and/or pixel shaders with the
330     // shader debugger. Debugging vertex shaders requires either REF or software
        vertex
331     // processing, and debugging pixel shaders requires REF.  The
332     // D3DXSHADER_FORCE_*_SOFTWARE_NOOPT flag improves the debug experience in the
333     // shader debugger.  It enables source level debugging, prevents instruction
334     // reordering, prevents dead code elimination, and forces the compiler to compile
```

```
335     // against the next higher available software target, which ensures that the
336     // unoptimized shaders do not exceed the shader model limitations.  Setting these
337     // flags will cause slower rendering since the shaders will be unoptimized and
338     // forced into software.  See the DirectX documentation for more information about
339     // using the shader debugger.
340     DWORD dwShaderFlags = D3DXFX_NOT_CLONEABLE;
341     #ifdef DEBUG_VS
342         dwShaderFlags |= D3DXSHADER_FORCE_VS_SOFTWARE_NOOPT;
343     #endif
344     #ifdef DEBUG_PS
345         dwShaderFlags |= D3DXSHADER_FORCE_PS_SOFTWARE_NOOPT;
346     #endif
347
348     // Read the D3DX effect file
349     WCHAR str[MAX_PATH];
350     V_RETURN(DXUTFindDXSDKMediaFileCch( str, MAX_PATH, L"RNAVis.fx" ));
351
352     // If this fails, there should be debug output as to
353     // they the .fx file failed to compile
354     //V_RETURN(D3DXCreateEffectFromFile(pd3dDevice, str, NULL, NULL, dwShaderFlags,
355     //                                  NULL, &g_pEffect, NULL));
356
357     // Setup the camera's view parameters
358 //    D3DXVECTOR3 vecEye(0.0f, 0.0f, -5.0f);
359 //    D3DXVECTOR3 vecAt (0.0f, 0.0f, -0.0f);
360 //    g_Camera.SetViewParams(&vecEye, &vecAt);
361
362     return S_OK;
363 }
364
365
366 //--------------------------------------------------------------------------------
    ---
367 // This callback function will be called immediately after the Direct3D device has been
368 // reset, which will happen after a lost device scenario. This is the best location to
369 // create D3DPOOL_DEFAULT resources since these resources need to be reloaded whenever
370 // the device is lost. Resources created here should be released in the OnLostDevice
371 // callback.
372 //--------------------------------------------------------------------------------
    ---
373 HRESULT CALLBACK OnResetDevice(IDirect3DDevice9* pd3dDevice,
374                                const D3DSURFACE_DESC* pBackBufferSurfaceDesc, void*
    pUserContext)
375 {
376     HRESULT hr;
377
378     V_RETURN(g_DialogResourceManager.OnResetDevice());
379     V_RETURN(g_SettingsDlg.OnResetDevice());
380
381     if(g_pFont)
382         V_RETURN(g_pFont->OnResetDevice());
383     //if(g_pEffect)
384     //    V_RETURN(g_pEffect->OnResetDevice());
385
386     // Create a sprite to help batch calls when drawing many lines of text
387     V_RETURN(D3DXCreateSprite(pd3dDevice, &g_pTextSprite));
388
389     // Setup the camera's projection parameters
390 //    float fAspectRatio = pBackBufferSurfaceDesc->Width / (FLOAT)
    pBackBufferSurfaceDesc->Height;
391 //    g_Camera.SetProjParams(D3DX_PI/4, fAspectRatio, 0.1f, 1000.0f);
```

```
392 //    g_Camera.SetWindow(pBackBufferSurfaceDesc->Width, pBackBufferSurfaceDesc->
    Height);
393
394     g_HUD.SetLocation(pBackBufferSurfaceDesc->Width - 405, pBackBufferSurfaceDesc->
    Height - 30);
395     g_HUD.SetSize(400, 170);
396     g_SampleUI.SetLocation(pBackBufferSurfaceDesc->Width - 405, 0);
397     g_SampleUI.SetSize(375, 300);
398
399     // Release the stage object
400     if(g_pStage != NULL)
401     {
402         delete g_pStage;
403         g_pStage = NULL;
404     }
405     g_pStage = new Stage(pd3dDevice, 10.0f, 0.0f, -5.0f, 0.0f);
406     g_pStage->SetTextureFromBitmap(g_hbm);
407     UpdateStageLines();
408
409     return S_OK;
410 }
411
412
413 //--------------------------------------------------------------------------------
    ---
414 // This callback function will be called once at the beginning of every frame. This
    is the
415 // best location for your application to handle updates to the scene, but is not
416 // intended to contain actual rendering calls, which should instead be placed in the
417 // OnFrameRender callback.
418 //--------------------------------------------------------------------------------
    ---
419 void CALLBACK OnFrameMove(IDirect3DDevice9* pd3dDevice, double fTime, float
    fElapsedTime, void* pUserContext)
420 {
421     // Update the camera's position based on user input
422 //    g_Camera.FrameMove(fElapsedTime);
423 }
424
425 //--------------------------------------------------------------------------------
    ---
426 // This callback function will be called at the end of every frame to perform all the

427 // rendering calls for the scene, and it will also be called if the window needs to
    be
428 // repainted. After this function has returned, DXUT will call
429 // IDirect3DDevice9::Present to display the contents of the next buffer in the swap
    chain
430 //--------------------------------------------------------------------------------
    ---
431 void CALLBACK OnFrameRender(IDirect3DDevice9* pd3dDevice, double fTime, float
    fElapsedTime, void* pUserContext)
432 {
433     HRESULT hr;
434     //D3DXMATRIXA16 mWorld;
435     //D3DXMATRIXA16 mView;
436     //D3DXMATRIXA16 mProj;
437     //D3DXMATRIXA16 mWorldViewProjection;
438
439     // If the settings dialog is being shown, then
440     // render it instead of rendering the app's scene
441     if(g_SettingsDlg.IsActive())
442     {
443         g_SettingsDlg.OnRender(fElapsedTime);
444         return;
445     }
446
```

```
447     // Clear the render target and the zbuffer
448     //V(pd3dDevice->Clear(0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, D3DCOLOR_ARGB
        (0, 45, 50, 170), 1.0f, 0));
449     V(pd3dDevice->Clear(0L, NULL, D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER, D3DCOLOR_ARGB(0,
        66, 75, 121), 1.0f, 0L));
450
451     // Setup the light
452     D3DLIGHT9 light;
453     D3DXVECTOR3 vecLightDirUnnormalized(0.0f, -1.0f, 1.0f);
454     ZeroMemory( &light, sizeof(D3DLIGHT9) );
455     light.Type        = D3DLIGHT_DIRECTIONAL;
456     light.Diffuse.r   = 1.0f;
457     light.Diffuse.g   = 1.0f;
458     light.Diffuse.b   = 1.0f;
459     D3DXVec3Normalize( (D3DXVECTOR3*)&light.Direction, &vecLightDirUnnormalized );
460     light.Position.x   = 0.0f;
461     light.Position.y   = -1.0f;
462     light.Position.z   = 1.0f;
463     light.Range        = 1000.0f;
464     V(pd3dDevice->SetLight(0, &light));
465     V(pd3dDevice->LightEnable(0, TRUE));
466     V(pd3dDevice->SetRenderState(D3DRS_LIGHTING, TRUE));
467     V(pd3dDevice->SetRenderState(D3DRS_AMBIENT, D3DCOLOR_XRGB(60, 60, 60)));
468
469     // Render the scene
470     if(SUCCEEDED(pd3dDevice->BeginScene()))
471     {
472         // Get the projection & view matrix from the camera class
473         //mWorld = *g_Camera.GetWorldMatrix();
474         //mProj = *g_Camera.GetProjMatrix();
475         //mView = *g_Camera.GetViewMatrix();
476
477         //mWorldViewProjection = mWorld * mView * mProj;
478
479         // Update the effect's variables.  Instead of using strings, it would
480         // be more efficient to cache a handle to the parameter by calling
481         // ID3DXEffect::GetParameterByName
482         //V(g_pEffect->SetMatrix("g_mWorldViewProjection", &mWorldViewProjection));
483         //V(g_pEffect->SetMatrix("g_mWorld", &mWorld));
484         //V(g_pEffect->SetFloat("g_fTime", (float)fTime));
485
486         //Setup camera and perspective
487         SetupCamera(pd3dDevice);
488
489         //Now that the 3D camera is setup, render the 3D objects
490         Render3D(pd3dDevice);
491
492         DXUT_BeginPerfEvent(DXUT_PERFEVENTCOLOR, L"HUD / Stats"); // These events are
       to help PIX identify what the code is doing
493         RenderText();
494         V(g_HUD.OnRender(fElapsedTime));
495         V(g_SampleUI.OnRender(fElapsedTime));
496         DXUT_EndPerfEvent();
497
498         V(pd3dDevice->EndScene());
499
500         m_dwFrames++;
501     }
502 }
503
504 // This function sets up the Camera
505 void SetupCamera(IDirect3DDevice9* pd3dDevice)
506 {
507     //Here we will setup the camera.
508     //The camera has three settings: "Camera Position", "Look at Position" and "Up
        Direction"
509     D3DXMATRIX matView;
```

```
510     D3DXMatrixLookAtLH(&matView, &D3DXVECTOR3(0.0f, 5.0f, -15.0f),  //Camera Position
511                                  &D3DXVECTOR3(0.0f, 0.0f, 0.0f),    //Look At
        Position
512                                  &D3DXVECTOR3(0.0f, 1.0f, 0.0f));   //Up Direction
513     pd3dDevice->SetTransform(D3DTS_VIEW, &matView);
514
515     //Here we specify the field of view, aspect ratio and near and far clipping
        planes.
516     D3DXMATRIX matProj;
517     D3DXMatrixPerspectiveFovLH(&matProj, D3DX_PI/4, 1.0f, 1.0f, 2000.0f);
518     pd3dDevice->SetTransform(D3DTS_PROJECTION, &matProj);
519
520     //Make sure that the z-buffer and lighting are enabled
521     pd3dDevice->SetRenderState(D3DRS_ZENABLE, D3DZB_TRUE);
522     pd3dDevice->SetRenderState(D3DRS_LIGHTING, TRUE);
523 }
524
525 // Render the 3D Scene
526 void Render3D(IDirect3DDevice9* pd3dDevice)
527 {
528     D3DXMATRIX MatWorld; // The final world matrix
529     D3DXMATRIX MatScale;  // Matrix for scaling
530     D3DXMATRIX MatRot;   // Final rotation matrix, applied to pMatWorld.
531
532     // Left-To-Right order of matrix concatenation is important
533
534     // Apply Translation
535     D3DXMatrixTranslation(&MatWorld, g_pStage->m_xPos, g_pStage->m_yPos, g_pStage->
        m_zPos);
536
537     // Apply Rotations
538     if(g_pStage->m_fPitch || g_pStage->m_fYaw || g_pStage->m_fRoll) {
539         //D3DXMatrixIdentity(&MatRot);
540         D3DXMatrixRotationYawPitchRoll(&MatRot, D3DXToRadian(g_pStage->m_fYaw),
        D3DXToRadian(g_pStage->m_fPitch), D3DXToRadian(g_pStage->m_fRoll));
541         D3DXMatrixMultiply(&MatWorld, &MatRot, &MatWorld); // Apply matrix to world
        matrix.
542     }
543
544     // Apply Scaling
545     D3DXMatrixScaling(&MatScale, g_pStage->m_fScale, g_pStage->m_fScale, g_pStage->
        m_fScale);
546     D3DXMatrixMultiply(&MatWorld, &MatScale, &MatWorld);
547
548     //Render our objects
549     pd3dDevice->SetTransform(D3DTS_WORLD, &MatWorld);
550     m_dwTotalPolygons += g_pStage->Render();
551 }
552
553 //-----------------------------------------------------------------------------
        ---
554 // Render the help and statistics text. This function uses the ID3DXFont interface
        for
555 // efficient text rendering.
556 //-----------------------------------------------------------------------------
        ---
557 void RenderText()
558 {
559     // The helper object simply helps keep track of text position, and color
560     // and then it calls pFont->DrawText( m_pSprite, strMsg, -1, &rc, DT_NOCLIP,
        m_clr );
561     // If NULL is passed in as the sprite object, then it will work however the
562     // pFont->DrawText() will not be batched together.  Batching calls will improves
        performance.
563     CDXUTTextHelper txtHelper(g_pFont, g_pTextSprite, 15);
564
565     // Output statistics
```

```
566     txtHelper.Begin();
567     txtHelper.SetInsertionPos(5, 5);
568     txtHelper.SetForegroundColor(D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f));
569     DWORD dwDuration = (timeGetTime() - m_dwStartTime) / 1000;
570     if(dwDuration <= 0)
571     {
572         txtHelper.DrawTextLine(L"Calculating...");
573     }
574     else
575     {
576         // Draw help
577         const D3DSURFACE_DESC* pd3dsdBackBuffer = DXUTGetBackBufferSurfaceDesc();
578         if(g_bShowHelp)
579         {
580             txtHelper.SetInsertionPos(5, 5);
581             //txtHelper.SetForegroundColor(D3DXCOLOR(1.0f, 0.75f, 0.0f, 1.0f));
582             txtHelper.SetForegroundColor(D3DXCOLOR(1.0f, 1.0f, 0.0f, 1.0f));
583             txtHelper.DrawTextLine(L"Controls (F1 to hide):");
584 
585             txtHelper.SetInsertionPos(40, 25);
586             txtHelper.DrawTextLine( L"X Axis: Z,z\n"
587                                     L"Y Axis: X,x\n"
588                                     L"Z Axis: C,c");
589 
590             txtHelper.SetInsertionPos(140, 25);
591             txtHelper.DrawTextLine( L"Roll: A,a\n"
592                                     L"Pitch: S,s\n"
593                                     L"Yaw: D,d");
594 
595             txtHelper.SetInsertionPos(240, 25);
596             txtHelper.DrawTextLine( L"Stage: F,f\n"
597                                     L"Scale: +,-\n"
598                                     L"Wireframe: F8");
599 
600             txtHelper.SetInsertionPos(340, 25);
601             txtHelper.DrawTextLine( L"Panel 1: 1\n"
602                                     L"Panel 2: 2\n"
603                                     L"Panel 3: 3");
604 
605 
606             txtHelper.SetInsertionPos(440, 25);
607             txtHelper.DrawTextLine( L"Panel 4: 4\n"
608                                     L"Reset: Home\n"
609                                     L"Quit: ESC");
610 
611             //txtHelper.SetInsertionPos(5, 5);
612             txtHelper.SetInsertionPos(10, pd3dsdBackBuffer->Height - 35);
613             txtHelper.SetForegroundColor(D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f));
614             //txtHelper.DrawTextLine(L"Duration: %d seconds. Frames: %d. FPS: %d.
    Pitch: %f. Yaw: %f. Roll: %f. Scale: %f. Left: %d. Right: %d. X = %d. Y = %d.",
    dwDuration, m_dwFrames, (m_dwFrames / dwDuration), g_pStage->m_fPitch, g_pStage->
    m_fYaw, g_pStage->m_fRoll, g_pStage->m_fScale, m_nMouseLeft, m_nMouseRight,
    m_nMouseX, m_nMouseY);
615             wchar_t buffer[255];
616             swprintf(buffer, L"Duration: %d s Frames: %d FPS: %d Pitch: %f. Yaw: %f.
    Roll: %f. Scale: %f.", dwDuration, m_dwFrames, (m_dwFrames / dwDuration),
    g_pStage->m_fPitch, g_pStage->m_fYaw, g_pStage->m_fRoll, g_pStage->m_fScale);
617             txtHelper.DrawTextLine(buffer);
618 
619             swprintf(buffer, L"X: %f.  Y: %f.  Z: %f.", g_pStage->m_xPos, g_pStage->
    m_yPos, g_pStage->m_zPos);
620             txtHelper.DrawTextLine(buffer);
621 
622             //txtHelper.SetForegroundColor(D3DXCOLOR(1.0f, 1.0f, 0.0f, 1.0f));
623             //txtHelper.DrawTextLine(DXUTGetFrameStats());
624             //txtHelper.DrawTextLine(DXUTGetDeviceStats());
625         }
```

```
626         else
627         {
628             txtHelper.SetInsertionPos(5, 5);
629             txtHelper.SetForegroundColor(D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f));
630             txtHelper.DrawTextLine(L"Press F1 for help");
631         }
632     }
633 
634     txtHelper.End();
635 }
636 
637 
638 //--------------------------------------------------------------------------------
    ---
639 // Before handling window messages, DXUT passes incoming windows
640 // messages to the application through this callback function. If the application
    sets
641 // *pbNoFurtherProcessing to TRUE, then DXUT will not process this message.
642 //--------------------------------------------------------------------------------
    ---
643 LRESULT CALLBACK MsgProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam, bool*
    pbNoFurtherProcessing, void* pUserContext)
644 {
645     // Always allow dialog resource manager calls to handle global messages
646     // so GUI state is updated correctly
647     *pbNoFurtherProcessing = g_DialogResourceManager.MsgProc(hWnd, uMsg, wParam,
    lParam);
648     if(*pbNoFurtherProcessing)
649         return 0;
650 
651     if(g_SettingsDlg.IsActive())
652     {
653         g_SettingsDlg.MsgProc(hWnd, uMsg, wParam, lParam);
654         return 0;
655     }
656 
657     // Give the dialogs a chance to handle the message first
658     *pbNoFurtherProcessing = g_HUD.MsgProc(hWnd, uMsg, wParam, lParam);
659     if(*pbNoFurtherProcessing)
660         return 0;
661     *pbNoFurtherProcessing = g_SampleUI.MsgProc(hWnd, uMsg, wParam, lParam);
662     if(*pbNoFurtherProcessing)
663         return 0;
664 
665     // Pass all remaining windows messages to camera so it can respond to user input
666 //    g_Camera.HandleMessages(hWnd, uMsg, wParam, lParam);
667 
668     switch( uMsg )
669     {
670         //
671         // Use WM_CHAR to handle parameter adjustment so
672         // that we can control the granularity based on
673         // the letter cases.
674         case WM_CHAR:
675         {
676             switch( wParam )
677             {
678                 case 'A':
679                 case 'a':
680                     if( 'a' == wParam )
681                         g_pStage->m_fRoll -= 0.5f;
682                     else
683                         g_pStage->m_fRoll += 0.5f;
684                     break;
685                 case 'S':
686                 case 's':
687                     if( 's' == wParam )
```

```
688                         g_pStage->m_fPitch -= 0.5f;
689                     else
690                         g_pStage->m_fPitch += 0.5f;
691                     break;
692                 case 'D':
693                 case 'd':
694                     if( 'd' == wParam)
695                         g_pStage->m_fYaw -= 0.5f;
696                     else
697                         g_pStage->m_fYaw += 0.5f;
698                     break;
699                 case 'F':
700                 case 'f':
701                     if( 'f' == wParam )
702                         g_pStage->SetStageAngle(g_pStage->m_fTheta -= 0.005f);
703                     else
704                         g_pStage->SetStageAngle(g_pStage->m_fTheta += 0.005f);
705                     break;
706                 case 'Z':
707                 case 'z':
708                     if( 'z' == wParam )
709                         g_pStage->m_xPos -= 0.5f;
710                     else
711                         g_pStage->m_xPos += 0.5f;
712                     break;
713                 case 'X':
714                 case 'x':
715                     if( 'x' == wParam )
716                         g_pStage->m_yPos -= 0.5f;
717                     else
718                         g_pStage->m_yPos += 0.5f;
719                     break;
720                 case 'C':
721                 case 'c':
722                     if( 'c' == wParam )
723                         g_pStage->m_zPos -= 0.5f;
724                     else
725                         g_pStage->m_zPos += 0.5f;
726                     break;
727                 case '_':
728                 case '-':
729                     g_pStage->m_fScale -= 0.5f;
730                     if(g_pStage->m_fScale < 1.0f) // Set lower bound on scaling
731                         g_pStage->m_fScale = 1.0f;
732                     break;
733                 case '+':
734                 case '=':
735                     g_pStage->m_fScale += 0.5f;
736                     if(g_pStage->m_fScale > 20.0f) // Set upper bound on scaling
737                         g_pStage->m_fScale = 20.0f;
738                     break;
739                 case '1':
740                 case '!': // Show Matrix Panel
741                     g_pStage->m_xPos = 0.0f;
742                     g_pStage->m_yPos = -7.0f;
743                     g_pStage->m_zPos = 0.0f;
744                     g_pStage->m_fRoll = 0.0f;
745                     g_pStage->m_fPitch = -72.0f;
746                     g_pStage->m_fYaw = 0.0f;
747                     g_pStage->m_fScale = 1.0f;
748                     g_pStage->SetStageAngle(D3DXToRadian(10.0f));
749                     break;
750                 case '2':
751                 case '@': // Show Planar Graph Panel
752                     g_pStage->m_xPos = 0.0f;
753                     g_pStage->m_yPos = -0.5f;
754                     g_pStage->m_zPos = -15.0f;
```

```
755                     g_pStage->m_fRoll = 0.0f;
756                     g_pStage->m_fPitch = 17.5f;
757                     g_pStage->m_fYaw = 0.0f;
758                     g_pStage->m_fScale = 1.0f;
759                     g_pStage->SetStageAngle(D3DXToRadian(10.0f));
760                     break;
761                 case '3':
762                 case '#': // Show Circular Graph Panel
763                     g_pStage->m_xPos = 6.5f;
764                     g_pStage->m_yPos = -3.5f;
765                     g_pStage->m_zPos = -9.0f;
766                     g_pStage->m_fRoll = -12.5f;
767                     g_pStage->m_fPitch = 2.0f;
768                     g_pStage->m_fYaw = -79.5f;
769                     g_pStage->m_fScale = 1.0f;
770                     g_pStage->SetStageAngle(D3DXToRadian(10.0f));
771                     break;
772                 case '4':
773                 case '$': // Show Bracketed Graph Panel
774                     g_pStage->m_xPos = -6.5f;
775                     g_pStage->m_yPos = -3.5f;
776                     g_pStage->m_zPos = -9.0f;
777                     g_pStage->m_fRoll = 12.5f;
778                     g_pStage->m_fPitch = 2.0f;
779                     g_pStage->m_fYaw = 79.5f;
780                     g_pStage->m_fScale = 1.0f;
781                     g_pStage->SetStageAngle(D3DXToRadian(10.0f));
782                     break;
783             }
784 
785             return 0;
786         }
787     }
788 
789     return 0;
790 }
791 
792 
793 //--------------------------------------------------------------------------------
    ---
794 // As a convenience, DXUT inspects the incoming windows messages for
795 // keystroke messages and decodes the message parameters to pass relevant keyboard
796 // messages to the application.  The framework does not remove the underlying
    keystroke
797 // messages, which are still passed to the application's MsgProc callback.
798 //--------------------------------------------------------------------------------
    ---
799 void CALLBACK KeyboardProc(UINT nChar, bool bKeyDown, bool bAltDown, void*
    pUserContext)
800 {
801     if(bKeyDown)
802     {
803         switch(nChar)
804         {
805             case VK_F1:
806                 g_bShowHelp = !g_bShowHelp;
807                 break;
808             case VK_HOME:
809                 g_pStage->m_xPos = 0.0f;
810                 g_pStage->m_yPos = -5.0f;
811                 g_pStage->m_zPos = 0.0f;
812                 g_pStage->m_fRoll = 0.0f;
813                 g_pStage->m_fPitch = 0.0f;
814                 g_pStage->m_fYaw = 0.0f;
815                 g_pStage->m_fScale = 1.0f;
816                 g_pStage->SetStageAngle(D3DXToRadian(10.0f));
817                 break;
```

```
818         }
819     }
820 }
821 
822 
823 //--------------------------------------------------------------------------------
    ---
824 // Handles the GUI events
825 //--------------------------------------------------------------------------------
    ---
826 void CALLBACK OnGUIEvent(UINT nEvent, int nControlID, CDXUTControl* pControl, void*
    pUserContext)
827 {
828     switch(nControlID)
829     {
830         case IDC_VISUALIZATIONOPTIONS:
831             {
832                 bool bWindowed = DXUTIsWindowed();
833                 if( !bWindowed )
834                     DXUTToggleFullScreen();
835 
836                 DialogBox(NULL, MAKEINTRESOURCE(IDD_VISUALIZATIONOPTIONS),
    DXUTGetHWND(), VisualizationOptions);
837 
838                 if( !bWindowed )
839                     DXUTToggleFullScreen();
840             }
841             break;
842         case IDC_TOGGLEFULLSCREEN:
843             DXUTToggleFullScreen();
844             break;
845         case IDC_CHANGEDEVICE:
846             g_SettingsDlg.SetActive(!g_SettingsDlg.IsActive());
847             break;
848         case IDC_NUSSINOVSTANDARD:
849             {
850                 bool bWindowed = DXUTIsWindowed();
851                 if( !bWindowed )
852                     DXUTToggleFullScreen();
853 
854                 DialogBox(NULL, MAKEINTRESOURCE(IDD_NUSSINOVSTANDARD), DXUTGetHWND(),
     NussinovStandard);
855 
856                 if( !bWindowed )
857                     DXUTToggleFullScreen();
858             }
859             break;
860         case IDC_NUSSINOVSCFG:
861             {
862                 bool bWindowed = DXUTIsWindowed();
863                 if( !bWindowed )
864                     DXUTToggleFullScreen();
865 
866                 DialogBox(NULL, MAKEINTRESOURCE(IDD_NUSSINOVSCFG), DXUTGetHWND(),
    NussinovSCFG);
867 
868                 if( !bWindowed )
869                     DXUTToggleFullScreen();
870             }
871             break;
872         case IDC_STEPFIRST:
873             g_pRNA->setStepPosition(1);
874             g_pNussinov->TraceBack(g_pRNA);  // Set the pairing array
875             g_pRNAGraphBMP->LoadFromRNASequence(g_pRNA->getTitle(), g_pRNA->
    getSequence(), g_pRNA->getStepPosition(), g_pRNA->getPairing());
876 
877             CreateDynamicTexture();
```

```
878             g_pStage->SetTextureFromBitmap(g_hbm);
879             UpdateStageLines();
880             break;
881         case IDC_STEPPREV:
882             if(g_pRNA->getStepPosition() > 1) {
883                 g_pRNA->setStepPosition(g_pRNA->getStepPosition() - 1);
884             }
885             g_pNussinov->TraceBack(g_pRNA);  // Set the pairing array
886             g_pRNAGraphBMP->LoadFromRNASequence(g_pRNA->getTitle(), g_pRNA->
    getSequence(), g_pRNA->getStepPosition(), g_pRNA->getPairing());
887
888             CreateDynamicTexture();
889             g_pStage->SetTextureFromBitmap(g_hbm);
890             UpdateStageLines();
891             break;
892         case IDC_STEPNEXT:
893             if(g_pRNA->getStepPosition() < g_pRNA->getLength()) {
894                 g_pRNA->setStepPosition(g_pRNA->getStepPosition() + 1);
895             }
896             g_pNussinov->TraceBack(g_pRNA);  // Set the pairing array
897             g_pRNAGraphBMP->LoadFromRNASequence(g_pRNA->getTitle(), g_pRNA->
    getSequence(), g_pRNA->getStepPosition(), g_pRNA->getPairing());
898
899             CreateDynamicTexture();
900             g_pStage->SetTextureFromBitmap(g_hbm);
901             UpdateStageLines();
902             break;
903         case IDC_STEPLAST:
904             g_pRNA->setStepPosition(g_pRNA->getLength());
905             g_pNussinov->TraceBack(g_pRNA);  // Set the pairing array
906             g_pRNAGraphBMP->LoadFromRNASequence(g_pRNA->getTitle(), g_pRNA->
    getSequence(), g_pRNA->getStepPosition(), g_pRNA->getPairing());
907
908             CreateDynamicTexture();
909             g_pStage->SetTextureFromBitmap(g_hbm);
910             UpdateStageLines();
911             break;
912     }
913 }
914
915
916 //--------------------------------------------------------------------------------
    ---
917 // This callback function will be called immediately after the Direct3D device has
918 // entered a lost state and before IDirect3DDevice9::Reset is called. Resources
    created
919 // in the OnResetDevice callback should be released here, which generally includes
    all
920 // D3DPOOL_DEFAULT resources. See the "Lost Devices" section of the documentation for

921 // information about lost devices.
922 //--------------------------------------------------------------------------------
    ---
923 void CALLBACK OnLostDevice(void* pUserContext)
924 {
925     g_DialogResourceManager.OnLostDevice();
926     g_SettingsDlg.OnLostDevice();
927     if(g_pFont)
928         g_pFont->OnLostDevice();
929     //if(g_pEffect)
930     //    g_pEffect->OnLostDevice();
931     SAFE_RELEASE(g_pTextSprite);
932
933     // Release the stage object
934     if(g_pStage != NULL)
935     {
936         delete g_pStage;
```

```
937         g_pStage = NULL;
938     }
939 }
940 
941 
942 //--------------------------------------------------------------------------------
    ---
943 // This callback function will be called immediately after the Direct3D device has
944 // been destroyed, which generally happens as a result of application termination or
945 // windowed/full screen toggles. Resources created in the OnCreateDevice callback
946 // should be released here, which generally includes all D3DPOOL_MANAGED resources.
947 //--------------------------------------------------------------------------------
    ---
948 void CALLBACK OnDestroyDevice(void* pUserContext)
949 {
950     g_DialogResourceManager.OnDestroyDevice();
951     g_SettingsDlg.OnDestroyDevice();
952     //SAFE_RELEASE(g_pEffect);
953     SAFE_RELEASE(g_pFont);
954 }
955 
956 // FUNCTION:  CreateDynamicTexure
957 // Used to dynamically create the texture that will be drawn
958 //
959 // @param   undefined   void
960 // @return  void
961 void CreateDynamicTexture()
962 {
963     HPEN hPen, hOldPen;
964     float fTextureSizeX, fTextureSizeY;
965     float fOriginX, fOriginY;
966 
967     HDC hdc = CreateCompatibleDC(NULL);
968     HBITMAP hbmOld = (HBITMAP)SelectObject(hdc, g_hbm);
969 
970     //BITMAP bm;
971     //HDC hdc = CreateCompatibleDC(NULL);
972     //HBITMAP hbmOld = (HBITMAP)SelectObject(hdc, g_hbm);
973     //GetObject(g_hbm, sizeof(bm), &bm);
974     //BitBlt(hdc, 0, 0, bm.bmWidth, bm.bmHeight, hdc, 0, 0, SRCCOPY);
975 
976 
977     //// Planar Graph
978     fTextureSizeX = 512.0f;
979     fTextureSizeY = 512.0f;
980     fOriginX = 0.0f;
981     fOriginY = 0.0f;
982     hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 255));
983     hOldPen = (HPEN)SelectObject(hdc, hPen);
984     Rectangle(hdc, (int)(fOriginX), (int)(fOriginY), (int)fTextureSizeX, (int)
    fTextureSizeY);
985     SelectObject(hdc, hOldPen);
986     DeleteObject(hPen);
987     g_pRNAGraphBMP->Draw(hdc, 1.0, fTextureSizeX, fTextureSizeY, g_MarkLoops,
    g_DrawBases, 0.5, g_LabelRate, g_DotPairs, 1, 1, 0.0, g_pRNA);
988 
989 
990 
991     // Circular Graph
992     fTextureSizeX = 512.0f;
993     fTextureSizeY = 512.0f;
994     fOriginX = 512.0f;
995     fOriginY = 0.0f;
996     hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 255));
997     hOldPen = (HPEN)SelectObject(hdc, hPen);
998     Rectangle(hdc, (int)(fOriginX), (int)(fOriginY), (int)(fOriginX + fTextureSizeX),
     (int)(fOriginY + fTextureSizeY));
```

```
 999     SelectObject(hdc, hOldPen);
1000     DeleteObject(hPen);
1001     SetMapMode(hdc, MM_ISOTROPIC);
1002     SetWindowExtEx(hdc, 1, 1, NULL);
1003     SetViewportExtEx(hdc, 1, -1, NULL);
1004     SetViewportOrgEx(hdc, (int)(fOriginX + (fTextureSizeX / 2.0f) + 0.5f), (int)
         (fOriginY + (fTextureSizeY / 2.0f) + 0.5f), NULL);
1005     g_pCircular->Draw(hdc, g_pRNA->getSequence(), g_pRNA->getStepPosition(), g_pRNA->
         getPairing(), fTextureSizeX, fTextureSizeY);
1006
1007
1008
1009     // Matrix Graph
1010     fTextureSizeX = 512.0f;
1011     fTextureSizeY = 512.0f;
1012     fOriginX = 0.0f;
1013     fOriginY = 512.0f;
1014     SetMapMode(hdc, MM_ISOTROPIC);
1015     SetWindowExtEx(hdc, 1, 1, NULL);
1016     SetViewportExtEx(hdc, 1, 1, NULL);
1017     SetViewportOrgEx(hdc, (int)(fOriginX), (int)(fOriginY), NULL);
1018     hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 255));
1019     hOldPen = (HPEN)SelectObject(hdc, hPen);
1020     Rectangle(hdc, 0, 0, (int)(fTextureSizeX), (int)(fTextureSizeY));
1021     SelectObject(hdc, hOldPen);
1022     DeleteObject(hPen);
1023     g_pMatrix->Draw(hdc, g_pRNA, g_pNussinov, fTextureSizeX, fTextureSizeY);
1024
1025
1026
1027     // Bracketed Graph
1028     fTextureSizeX = 512.0f;
1029     fTextureSizeY = 512.0f;
1030     fOriginX = 512.0f;
1031     fOriginY = 512.0f;
1032     SetMapMode(hdc, MM_ISOTROPIC);
1033     SetWindowExtEx(hdc, 1, 1, NULL);
1034     SetViewportExtEx(hdc, 1, 1, NULL);
1035     SetViewportOrgEx(hdc, (int)(fOriginX), (int)(fOriginY), NULL);
1036     hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 255));
1037     hOldPen = (HPEN)SelectObject(hdc, hPen);
1038     Rectangle(hdc, 0, 0, (int)fTextureSizeX, (int)fTextureSizeY);
1039     SelectObject(hdc, hOldPen);
1040     DeleteObject(hPen);
1041     g_pBracketed->Draw(hdc, g_pRNA, g_pNussinov, fTextureSizeX, fTextureSizeY);
1042
1043
1044
1045     // Progress Graph
1046     fTextureSizeX = 1024.0f;
1047     fTextureSizeY = 128.0f;
1048     fOriginX = 0.0f;
1049     fOriginY = 1152.0f;
1050     SetMapMode(hdc, MM_ISOTROPIC);
1051     SetWindowExtEx(hdc, 1, 1, NULL);
1052     SetViewportExtEx(hdc, 1, 1, NULL);
1053     SetViewportOrgEx(hdc, (int)(fOriginX), (int)(fOriginY), NULL);
1054     g_pProgress->Draw(hdc, g_pRNA->getSequence(), g_pRNA->getLength(), g_pRNA->
         getStepPosition(), fTextureSizeX, fTextureSizeY);
1055
1056
1057     SelectObject(hdc, hbmOld);
1058     DeleteDC(hdc);
1059 }
1060
1061 // Message handler for Visualization Options
1062 INT_PTR CALLBACK VisualizationOptions(HWND hDlg, UINT message, WPARAM wParam, LPARAM
```

```
    lParam)
1063 {
1064     UNREFERENCED_PARAMETER(lParam);
1065     switch (message)
1066     {
1067     case WM_INITDIALOG:
1068         // Initialize the Draw Lines Check Box
1069         if(g_DrawLines)
1070             SendDlgItemMessage(hDlg, IDC_DRAW3DLINES, BM_SETCHECK, BST_CHECKED,      ↙
    (LPARAM)0);
1071         else
1072             SendDlgItemMessage(hDlg, IDC_DRAW3DLINES, BM_SETCHECK, BST_UNCHECKED,    ↙
    (LPARAM)0);
1073
1074         // Initialize the Mark Loops Check Box
1075         if(g_MarkLoops)
1076             SendDlgItemMessage(hDlg, IDC_MARKLOOPS, BM_SETCHECK, BST_CHECKED,        ↙
    (LPARAM)0);
1077         else
1078             SendDlgItemMessage(hDlg, IDC_MARKLOOPS, BM_SETCHECK, BST_UNCHECKED,      ↙
    (LPARAM)0);
1079
1080         // Initialize the Draw Bases Check Box
1081         if(g_DrawBases)
1082             SendDlgItemMessage(hDlg, IDC_DRAWBASES, BM_SETCHECK, BST_CHECKED,        ↙
    (LPARAM)0);
1083         else
1084             SendDlgItemMessage(hDlg, IDC_DRAWBASES, BM_SETCHECK, BST_UNCHECKED,      ↙
    (LPARAM)0);
1085
1086         // Initialize the Dot Pairs Check Box
1087         if(g_DotPairs)
1088             SendDlgItemMessage(hDlg, IDC_DOTPAIRS, BM_SETCHECK, BST_CHECKED, (LPARAM)↙
    0);
1089         else
1090             SendDlgItemMessage(hDlg, IDC_DOTPAIRS, BM_SETCHECK, BST_UNCHECKED,       ↙
    (LPARAM)0);
1091
1092         SetDlgItemInt(hDlg, IDC_LABELRATE, g_LabelRate, true);  // Initialize Label  ↙
    Rate
1093
1094         return (INT_PTR)TRUE;
1095
1096     case WM_COMMAND:
1097         switch (LOWORD(wParam))
1098         {
1099         case IDOK:
1100             {
1101                 if(SendDlgItemMessageA(hDlg, IDC_DRAW3DLINES, BM_GETCHECK, (WPARAM)0,↙
     (LPARAM)0) == BST_CHECKED)
1102                     g_DrawLines = true;
1103                 else
1104                     g_DrawLines = false;
1105                 if(SendDlgItemMessageA(hDlg, IDC_MARKLOOPS, BM_GETCHECK, (WPARAM)0,  ↙
    (LPARAM)0) == BST_CHECKED)
1106                     g_MarkLoops = true;
1107                 else
1108                     g_MarkLoops = false;
1109                 if(SendDlgItemMessageA(hDlg, IDC_DRAWBASES, BM_GETCHECK, (WPARAM)0,  ↙
    (LPARAM)0) == BST_CHECKED)
1110                     g_DrawBases = true;
1111                 else
1112                     g_DrawBases = false;
1113                 if(SendDlgItemMessageA(hDlg, IDC_DOTPAIRS, BM_GETCHECK, (WPARAM)0,   ↙
    (LPARAM)0) == BST_CHECKED)
1114                     g_DotPairs = true;
1115                 else
```

```
1116                    g_DotPairs = false;
1117
1118                // Get Label Rate from the text box
1119                BOOL success;
1120                g_LabelRate = GetDlgItemInt(hDlg, IDC_LABELRATE, &success, true);
1121                if(!success)
1122                {
1123                    MessageBoxA(hDlg, "The Label Rate Is Invalid", "Invalid Value",
    MB_OK);
1124                    break;
1125                }
1126
1127                CreateDynamicTexture();
1128                g_pStage->SetTextureFromBitmap(g_hbm);
1129                UpdateStageLines();
1130
1131                EndDialog(hDlg, LOWORD(wParam));
1132                return (INT_PTR)TRUE;
1133            }
1134            break;
1135        case IDCANCEL:
1136            EndDialog(hDlg, LOWORD(wParam));
1137            return (INT_PTR)TRUE;
1138            break;
1139        }
1140        break;
1141    }
1142    return (INT_PTR)FALSE;
1143 }
1144
1145 // Message handler for RNAInputBox.
1146 INT_PTR CALLBACK NussinovStandard(HWND hDlg, UINT message, WPARAM wParam, LPARAM
    lParam)
1147 {
1148    UNREFERENCED_PARAMETER(lParam);
1149    switch (message)
1150    {
1151    case WM_INITDIALOG:
1152        // Initialize Title
1153        SetDlgItemTextA(hDlg, IDC_TITLE, g_pRNA->getTitle());
1154
1155        // Initialize Sequence
1156        SetDlgItemTextA(hDlg, IDC_SEQUENCE, g_pRNA->getSequence());
1157
1158        // Initialize the Scoring Matrix
1159        SetDlgItemInt(hDlg, IDC_AA, g_pNussinov->getScoringMatrix(0, 0), true);
1160        SetDlgItemInt(hDlg, IDC_AC, g_pNussinov->getScoringMatrix(0, 1), true);
1161        SetDlgItemInt(hDlg, IDC_AG, g_pNussinov->getScoringMatrix(0, 2), true);
1162        SetDlgItemInt(hDlg, IDC_AU, g_pNussinov->getScoringMatrix(0, 3), true);
1163        SetDlgItemInt(hDlg, IDC_CA, g_pNussinov->getScoringMatrix(1, 0), true);
1164        SetDlgItemInt(hDlg, IDC_CC, g_pNussinov->getScoringMatrix(1, 1), true);
1165        SetDlgItemInt(hDlg, IDC_CG, g_pNussinov->getScoringMatrix(1, 2), true);
1166        SetDlgItemInt(hDlg, IDC_CU, g_pNussinov->getScoringMatrix(1, 3), true);
1167        SetDlgItemInt(hDlg, IDC_GA, g_pNussinov->getScoringMatrix(2, 0), true);
1168        SetDlgItemInt(hDlg, IDC_GC, g_pNussinov->getScoringMatrix(2, 1), true);
1169        SetDlgItemInt(hDlg, IDC_GG, g_pNussinov->getScoringMatrix(2, 2), true);
1170        SetDlgItemInt(hDlg, IDC_GU, g_pNussinov->getScoringMatrix(2, 3), true);
1171        SetDlgItemInt(hDlg, IDC_UA, g_pNussinov->getScoringMatrix(3, 0), true);
1172        SetDlgItemInt(hDlg, IDC_UC, g_pNussinov->getScoringMatrix(3, 1), true);
1173        SetDlgItemInt(hDlg, IDC_UG, g_pNussinov->getScoringMatrix(3, 2), true);
1174        SetDlgItemInt(hDlg, IDC_UU, g_pNussinov->getScoringMatrix(3, 3), true);
1175
1176        // Initiallize Hairpin Loop Length
1177        SetDlgItemInt(hDlg, IDC_HAIRPINLENGTH, g_pNussinov->getMinHairpinLength(),
    true); // Set default minimum hairpin length
1178
1179        return (INT_PTR)TRUE;
```

```
1180
1181        case WM_COMMAND:
1182            switch (LOWORD(wParam))
1183            {
1184            case IDOK:
1185                {
1186                    // Get the Title from the text box
1187                    int TitleLen = (int) SendDlgItemMessageA(hDlg, IDC_TITLE,
    EM_LINELENGTH, (WPARAM)0, (LPARAM)0); // Get number of characters.
1188                    char* szTitleBuffer = new char[TitleLen + 1];
1189                    GetDlgItemTextA(hDlg, IDC_TITLE, szTitleBuffer, TitleLen + 1); // Get
     RNA Sequence from edit box
1190
1191                    // Get the RNA Sequence from the text box
1192                    int lineCount = SendDlgItemMessageA(hDlg, IDC_SEQUENCE,
    EM_GETLINECOUNT, (WPARAM)0, (LPARAM)0); //Get number of lines in edit box
1193                    int lineOffset;
1194                    int RNALen = 0;
1195                    for(int i = 0; i < lineCount; i++)
1196                    {
1197                        lineOffset = SendDlgItemMessageA(hDlg, IDC_SEQUENCE, EM_LINEINDEX
    , (WPARAM)i, (LPARAM)0);
1198                        RNALen += (int)SendDlgItemMessageA(hDlg, IDC_SEQUENCE,
    EM_LINELENGTH, (WPARAM)lineOffset, (LPARAM)0) + 1;
1199                    }
1200                    //int RNALen = (int) SendDlgItemMessageA(hDlg, IDC_SEQUENCE,
    EM_LINELENGTH, (WPARAM) 0, (LPARAM) 0); // Get number of characters.
1201                    char* szRNABuffer = new char[RNALen + 1];
1202                    GetDlgItemTextA(hDlg, IDC_SEQUENCE, szRNABuffer, RNALen + 1); // Get
    RNA Sequence from edit box
1203
1204                    // Get Scoring Matrix from the text boxes
1205                    BOOL success;
1206                    int AA, AC, AG, AU, CA, CC, CG, CU, GA, GC, GG, GU, UA, UC, UG, UU;
1207                    AA = GetDlgItemInt(hDlg, IDC_AA, &success, true);
1208                    if(!success)
1209                    {
1210                        MessageBoxA(hDlg, "Scoring Matrix Has Invalid Data In Row: A,
    Column: A", "Invalid Scoring Matrix Entry", MB_OK);
1211                        break;
1212                    }
1213                    AC = GetDlgItemInt(hDlg, IDC_AC, &success, true);
1214                    if(!success)
1215                    {
1216                        MessageBoxA(hDlg, "Scoring Matrix Has Invalid Data In Row: A,
    Column: C", "Invalid Scoring Matrix Entry", MB_OK);
1217                        break;
1218                    }
1219                    AG = GetDlgItemInt(hDlg, IDC_AG, &success, true);
1220                    if(!success)
1221                    {
1222                        MessageBoxA(hDlg, "Scoring Matrix Has Invalid Data In Row: A,
    Column: G", "Invalid Scoring Matrix Entry", MB_OK);
1223                        break;
1224                    }
1225                    AU = GetDlgItemInt(hDlg, IDC_AU, &success, true);
1226                    if(!success)
1227                    {
1228                        MessageBoxA(hDlg, "Scoring Matrix Has Invalid Data In Row: A,
    Column: U", "Invalid Scoring Matrix Entry", MB_OK);
1229                        break;
1230                    }
1231                    CA = GetDlgItemInt(hDlg, IDC_CA, &success, true);
1232                    if(!success)
1233                    {
1234                        MessageBoxA(hDlg, "Scoring Matrix Has Invalid Data In Row: C,
    Column: A", "Invalid Scoring Matrix Entry", MB_OK);
```

```
1235                    break;
1236                }
1237                CC = GetDlgItemInt(hDlg, IDC_CC, &success, true);
1238                if(!success)
1239                {
1240                    MessageBoxA(hDlg, "Scoring Matrix Has Invalid Data In Row: C,
    Column: C", "Invalid Scoring Matrix Entry", MB_OK);
1241                    break;
1242                }
1243                CG = GetDlgItemInt(hDlg, IDC_CG, &success, true);
1244                if(!success)
1245                {
1246                    MessageBoxA(hDlg, "Scoring Matrix Has Invalid Data In Row: C,
    Column: G", "Invalid Scoring Matrix Entry", MB_OK);
1247                    break;
1248                }
1249                CU = GetDlgItemInt(hDlg, IDC_CU, &success, true);
1250                if(!success)
1251                {
1252                    MessageBoxA(hDlg, "Scoring Matrix Has Invalid Data In Row: C,
    Column: U", "Invalid Scoring Matrix Entry", MB_OK);
1253                    break;
1254                }
1255                GA = GetDlgItemInt(hDlg, IDC_GA, &success, true);
1256                if(!success)
1257                {
1258                    MessageBoxA(hDlg, "Scoring Matrix Has Invalid Data In Row: G,
    Column: A", "Invalid Scoring Matrix Entry", MB_OK);
1259                    break;
1260                }
1261                GC = GetDlgItemInt(hDlg, IDC_GC, &success, true);
1262                if(!success)
1263                {
1264                    MessageBoxA(hDlg, "Scoring Matrix Has Invalid Data In Row: G,
    Column: C", "Invalid Scoring Matrix Entry", MB_OK);
1265                    break;
1266                }
1267                GG = GetDlgItemInt(hDlg, IDC_GG, &success, true);
1268                if(!success)
1269                {
1270                    MessageBoxA(hDlg, "Scoring Matrix Has Invalid Data In Row: G,
    Column: G", "Invalid Scoring Matrix Entry", MB_OK);
1271                    break;
1272                }
1273                GU = GetDlgItemInt(hDlg, IDC_GU, &success, true);
1274                if(!success)
1275                {
1276                    MessageBoxA(hDlg, "Scoring Matrix Has Invalid Data In Row: G,
    Column: U", "Invalid Scoring Matrix Entry", MB_OK);
1277                    break;
1278                }
1279                UA = GetDlgItemInt(hDlg, IDC_UA, &success, true);
1280                if(!success)
1281                {
1282                    MessageBoxA(hDlg, "Scoring Matrix Has Invalid Data In Row: U,
    Column: A", "Invalid Scoring Matrix Entry", MB_OK);
1283                    break;
1284                }
1285                UC = GetDlgItemInt(hDlg, IDC_UC, &success, true);
1286                if(!success)
1287                {
1288                    MessageBoxA(hDlg, "Scoring Matrix Has Invalid Data In Row: U,
    Column: C", "Invalid Scoring Matrix Entry", MB_OK);
1289                    break;
1290                }
1291                UG = GetDlgItemInt(hDlg, IDC_UG, &success, true);
1292                if(!success)
```

```
1293                {
1294                    MessageBoxA(hDlg, "Scoring Matrix Has Invalid Data In Row: U, Column: G", "Invalid Scoring Matrix Entry", MB_OK);
1295                    break;
1296                }
1297                UU = GetDlgItemInt(hDlg, IDC_UU, &success, true);
1298                if(!success)
1299                {
1300                    MessageBoxA(hDlg, "Scoring Matrix Has Invalid Data In Row: U, Column: U", "Invalid Scoring Matrix Entry", MB_OK);
1301                    break;
1302                }
1303
1304                // Get Scoring Matrix from the text boxes
1305                int minHairpin;
1306                minHairpin = GetDlgItemInt(hDlg, IDC_HAIRPINLENGTH, &success, true);
1307                if(!success)
1308                {
1309                    MessageBoxA(hDlg, "The Minimum Hairpin Length Is Invalid", "Invalid Value", MB_OK);
1310                    break;
1311                }
1312
1313                checkFASTA(szRNABuffer); // If FASTA sequence then remove extra text and leave sequence
1314                int lengthRNA = checkRNA(szRNABuffer); // Verify correct RNA symbols have been entered
1315
1316                delete g_pRNA;
1317                g_pRNA = new RNA(szTitleBuffer, szRNABuffer, lengthRNA); // Instantiate the RNA class object
1318
1319                delete g_pNussinov;
1320                g_pNussinov = new Nussinov(Nussinov::NussinovType::NussinovStandard, g_pRNA->getSequence(), g_pRNA->getLength());  // Instantiate an Nussinov class object
1321                g_pNussinov->setScoringMatrix(AA, AC, AG, AU, CA, CC, CG, CU, GA, GC, GG, GU, UA, UC, UG, UU); // Set scoring matrix
1322                g_pNussinov->setMinHairpinLength(minHairpin); // Set the minimum hairpin loop length
1323                g_pNussinov->FillStage();
1324                g_pNussinov->TraceBack(g_pRNA);  // Set the pairing array
1325
1326                delete g_pRNAGraphBMP;
1327                g_pRNAGraphBMP = new RNAGraphBMP(0.7f);  // Instantiate the RNAGraphBMP class object
1328                g_pRNAGraphBMP->LoadFromRNASequence(g_pRNA->getTitle(), g_pRNA->getSequence(), g_pRNA->getStepPosition(), g_pRNA->getPairing());
1329
1330                delete g_pCircular;
1331                g_pCircular = new CircularGraph;  // Instantiate the CircularGraph class object
1332                delete g_pMatrix;
1333                g_pMatrix = new MatrixGraph;  // Instantiate the MatrixGraph class object
1334                delete g_pProgress;
1335                g_pProgress = new ProgressGraph; // Instantiate the ProgressGraph class object
1336                delete g_pBracketed;
1337                g_pBracketed = new BracketedGraph; // Instantiate the BracketedGraph class object
1338
1339                CreateDynamicTexture();
1340                g_pStage->SetTextureFromBitmap(g_hbm);
1341                UpdateStageLines();
1342
1343                EndDialog(hDlg, LOWORD(wParam));
```

```
1344                return (INT_PTR)TRUE;
1345            }
1346            break;
1347        case IDCANCEL:
1348            EndDialog(hDlg, LOWORD(wParam));
1349            return (INT_PTR)TRUE;
1350            break;
1351        case IDC_STANDARD:
1352            // Initialize standard Scoring Matrix
1353            SetDlgItemInt(hDlg, IDC_AA, 0, true);
1354            SetDlgItemInt(hDlg, IDC_AC, 0, true);
1355            SetDlgItemInt(hDlg, IDC_AG, 0, true);
1356            SetDlgItemInt(hDlg, IDC_AU, 1, true);
1357            SetDlgItemInt(hDlg, IDC_CA, 0, true);
1358            SetDlgItemInt(hDlg, IDC_CC, 0, true);
1359            SetDlgItemInt(hDlg, IDC_CG, 1, true);
1360            SetDlgItemInt(hDlg, IDC_CU, 0, true);
1361            SetDlgItemInt(hDlg, IDC_GA, 0, true);
1362            SetDlgItemInt(hDlg, IDC_GC, 1, true);
1363            SetDlgItemInt(hDlg, IDC_GG, 0, true);
1364            SetDlgItemInt(hDlg, IDC_GU, 0, true);
1365            SetDlgItemInt(hDlg, IDC_UA, 1, true);
1366            SetDlgItemInt(hDlg, IDC_UC, 0, true);
1367            SetDlgItemInt(hDlg, IDC_UG, 0, true);
1368            SetDlgItemInt(hDlg, IDC_UU, 0, true);
1369            break;
1370        case IDC_STANDARDGU:
1371            // Initialize standard plus GU Scoring Matrix
1372            SetDlgItemInt(hDlg, IDC_AA, 0, true);
1373            SetDlgItemInt(hDlg, IDC_AC, 0, true);
1374            SetDlgItemInt(hDlg, IDC_AG, 0, true);
1375            SetDlgItemInt(hDlg, IDC_AU, 1, true);
1376            SetDlgItemInt(hDlg, IDC_CA, 0, true);
1377            SetDlgItemInt(hDlg, IDC_CC, 0, true);
1378            SetDlgItemInt(hDlg, IDC_CG, 1, true);
1379            SetDlgItemInt(hDlg, IDC_CU, 0, true);
1380            SetDlgItemInt(hDlg, IDC_GA, 0, true);
1381            SetDlgItemInt(hDlg, IDC_GC, 1, true);
1382            SetDlgItemInt(hDlg, IDC_GG, 0, true);
1383            SetDlgItemInt(hDlg, IDC_GU, 1, true);
1384            SetDlgItemInt(hDlg, IDC_UA, 1, true);
1385            SetDlgItemInt(hDlg, IDC_UC, 0, true);
1386            SetDlgItemInt(hDlg, IDC_UG, 1, true);
1387            SetDlgItemInt(hDlg, IDC_UU, 0, true);
1388            break;
1389        case IDC_COMPLEX:
1390            // Initialize standard Scoring Matrix
1391            SetDlgItemInt(hDlg, IDC_AA, 0, true);
1392            SetDlgItemInt(hDlg, IDC_AC, 0, true);
1393            SetDlgItemInt(hDlg, IDC_AG, 0, true);
1394            SetDlgItemInt(hDlg, IDC_AU, 2, true);
1395            SetDlgItemInt(hDlg, IDC_CA, 0, true);
1396            SetDlgItemInt(hDlg, IDC_CC, 0, true);
1397            SetDlgItemInt(hDlg, IDC_CG, 3, true);
1398            SetDlgItemInt(hDlg, IDC_CU, 0, true);
1399            SetDlgItemInt(hDlg, IDC_GA, 0, true);
1400            SetDlgItemInt(hDlg, IDC_GC, 3, true);
1401            SetDlgItemInt(hDlg, IDC_GG, 0, true);
1402            SetDlgItemInt(hDlg, IDC_GU, 1, true);
1403            SetDlgItemInt(hDlg, IDC_UA, 2, true);
1404            SetDlgItemInt(hDlg, IDC_UC, 0, true);
1405            SetDlgItemInt(hDlg, IDC_UG, 1, true);
1406            SetDlgItemInt(hDlg, IDC_UU, 0, true);
1407            break;
1408        }
1409        break;
1410    }
```

```
1411        return (INT_PTR)FALSE;
1412    }
1413
1414    // Message handler for about box.
1415    INT_PTR CALLBACK NussinovSCFG(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
1416    {
1417        UNREFERENCED_PARAMETER(lParam);
1418        switch (message)
1419        {
1420        case WM_INITDIALOG:
1421            // Initialize Title
1422            SetDlgItemTextA(hDlg, IDC_TITLE, g_pRNA->getTitle());
1423
1424            // Initialize Sequence
1425            SetDlgItemTextA(hDlg, IDC_SEQUENCE, g_pRNA->getSequence());
1426
1427            // Initialize the Scoring Matrix
1428            char strNumber[64];
1429            sprintf(strNumber, "%f", exp(g_pNussinov->getProbMatrix(0, 0)));
1430            SetDlgItemTextA(hDlg, IDC_AS, strNumber);
1431            sprintf(strNumber, "%f", exp(g_pNussinov->getProbMatrix(1, 0)));
1432            SetDlgItemTextA(hDlg, IDC_CS, strNumber);
1433            sprintf(strNumber, "%f", exp(g_pNussinov->getProbMatrix(2, 0)));
1434            SetDlgItemTextA(hDlg, IDC_GS, strNumber);
1435            sprintf(strNumber, "%f", exp(g_pNussinov->getProbMatrix(3, 0)));
1436            SetDlgItemTextA(hDlg, IDC_US, strNumber);
1437            sprintf(strNumber, "%f", exp(g_pNussinov->getProbMatrix(0, 1)));
1438            SetDlgItemTextA(hDlg, IDC_SA, strNumber);
1439            sprintf(strNumber, "%f", exp(g_pNussinov->getProbMatrix(1, 1)));
1440            SetDlgItemTextA(hDlg, IDC_SC, strNumber);
1441            sprintf(strNumber, "%f", exp(g_pNussinov->getProbMatrix(2, 1)));
1442            SetDlgItemTextA(hDlg, IDC_SG, strNumber);
1443            sprintf(strNumber, "%f", exp(g_pNussinov->getProbMatrix(3, 1)));
1444            SetDlgItemTextA(hDlg, IDC_SU, strNumber);
1445            sprintf(strNumber, "%f", exp(g_pNussinov->getProbMatrix(0, 2)));
1446            SetDlgItemTextA(hDlg, IDC_ASU, strNumber);
1447            sprintf(strNumber, "%f", exp(g_pNussinov->getProbMatrix(1, 2)));
1448            SetDlgItemTextA(hDlg, IDC_CSG, strNumber);
1449            sprintf(strNumber, "%f", exp(g_pNussinov->getProbMatrix(2, 2)));
1450            SetDlgItemTextA(hDlg, IDC_GSC, strNumber);
1451            sprintf(strNumber, "%f", exp(g_pNussinov->getProbMatrix(3, 2)));
1452            SetDlgItemTextA(hDlg, IDC_USA, strNumber);
1453            sprintf(strNumber, "%f", exp(g_pNussinov->getProbSS()));
1454            SetDlgItemTextA(hDlg, IDC_SS, strNumber);
1455
1456            // Initiallize Hairpin Loop Length
1457            SetDlgItemInt(hDlg, IDC_HAIRPINLENGTH, g_pNussinov->getMinHairpinLength(),
        true); // Set default minimum hairpin length
1458
1459            return (INT_PTR)TRUE;
1460
1461        case WM_COMMAND:
1462            switch(LOWORD(wParam))
1463            {
1464            case IDOK:
1465                {
1466                    // Get the Title from the text box
1467                    int TitleLen = (int) SendDlgItemMessageA(hDlg, IDC_TITLE,
        EM_LINELENGTH, (WPARAM) 0, (LPARAM) 0); // Get number of characters.
1468                    char* szTitleBuffer = new char[TitleLen + 1];
1469                    GetDlgItemTextA(hDlg, IDC_TITLE, szTitleBuffer, TitleLen + 1); // Get
         RNA Sequence from edit box
1470
1471                    // Get the RNA Sequence from the text box
1472                    int lineCount = SendDlgItemMessageA(hDlg, IDC_SEQUENCE,
        EM_GETLINECOUNT, (WPARAM)0, (LPARAM)0); //Get number of lines in edit box
1473                    int lineOffset;
```

```
1474                int RNALen = 0;
1475                for(int i = 0; i < lineCount; i++)
1476                {
1477                    lineOffset = SendDlgItemMessageA(hDlg, IDC_SEQUENCE, EM_LINEINDEX
    , (WPARAM)i, (LPARAM)0);
1478                    RNALen += (int)SendDlgItemMessageA(hDlg, IDC_SEQUENCE,
    EM_LINELENGTH, (WPARAM)lineOffset, (LPARAM)0) + 1;
1479                }
1480                //int RNALen = (int) SendDlgItemMessageA(hDlg, IDC_SEQUENCE,
    EM_LINELENGTH, (WPARAM) 0, (LPARAM) 0); // Get number of characters.
1481                char* szRNABuffer = new char[RNALen + 1];
1482                GetDlgItemTextA(hDlg, IDC_SEQUENCE, szRNABuffer, RNALen + 1); // Get
    RNA Sequence from edit box
1483
1484                //atof(str);
1485                // Get Probability Matrix from the text boxes
1486                char strNumber[64];
1487                float aS, cS, gS, uS, Sa, Sc, Sg, Su, aSu, cSg, gSc, uSa, SS;
1488
1489                GetDlgItemTextA(hDlg, IDC_AS, strNumber, 63);
1490                aS = (float)atof(strNumber);
1491                GetDlgItemTextA(hDlg, IDC_CS, strNumber, 63);
1492                cS = (float)atof(strNumber);
1493                GetDlgItemTextA(hDlg, IDC_GS, strNumber, 63);
1494                gS = (float)atof(strNumber);
1495                GetDlgItemTextA(hDlg, IDC_US, strNumber, 63);
1496                uS = (float)atof(strNumber);
1497                GetDlgItemTextA(hDlg, IDC_SA, strNumber, 63);
1498                Sa = (float)atof(strNumber);
1499                GetDlgItemTextA(hDlg, IDC_SC, strNumber, 63);
1500                Sc = (float)atof(strNumber);
1501                GetDlgItemTextA(hDlg, IDC_SG, strNumber, 63);
1502                Sg = (float)atof(strNumber);
1503                GetDlgItemTextA(hDlg, IDC_SU, strNumber, 63);
1504                Su = (float)atof(strNumber);
1505                GetDlgItemTextA(hDlg, IDC_ASU, strNumber, 63);
1506                aSu = (float)atof(strNumber);
1507                GetDlgItemTextA(hDlg, IDC_CSG, strNumber, 63);
1508                cSg = (float)atof(strNumber);
1509                GetDlgItemTextA(hDlg, IDC_GSC, strNumber, 63);
1510                gSc = (float)atof(strNumber);
1511                GetDlgItemTextA(hDlg, IDC_USA, strNumber, 63);
1512                uSa = (float)atof(strNumber);
1513                GetDlgItemTextA(hDlg, IDC_SS, strNumber, 63);
1514                SS = (float)atof(strNumber);
1515
1516                // Get Scoring Matrix from the text boxes
1517                BOOL success;
1518                int minHairpin;
1519                minHairpin = GetDlgItemInt(hDlg, IDC_HAIRPINLENGTH, &success, true);
1520                if(!success)
1521                {
1522                    MessageBoxA(hDlg, "The Minimum Hairpin Length Is Invalid",
    "Invalid Value", MB_OK);
1523                    break;
1524                }
1525
1526                checkFASTA(szRNABuffer); // If FASTA sequence then remove extra text
    and leave sequence
1527                int lengthRNA = checkRNA(szRNABuffer); // Verify correct RNA symbols
    have been entered
1528
1529                delete g_pRNA;
1530                g_pRNA = new RNA(szTitleBuffer, szRNABuffer, lengthRNA); //
    Instantiate the RNA class object
1531
1532                delete g_pNussinov;
```

```
1533                g_pNussinov = new Nussinov(Nussinov::NussinovType::NussinovSCFG,
    g_pRNA->getSequence(), g_pRNA->getLength());  // Instantiate an Nussinov class
    object
1534                g_pNussinov->setProbMatrix(aS, cS, gS, uS, Sa, Sc, Sg, Su, aSu, cSg,
    gSc, uSa, SS); // Set scoring matrix
1535                g_pNussinov->setMinHairpinLength(minHairpin); // Set the minimum
    hairpin loop length
1536                g_pNussinov->FillStage();
1537                g_pNussinov->TraceBack(g_pRNA);
1538
1539                delete g_pRNAGraphBMP;
1540                g_pRNAGraphBMP = new RNAGraphBMP(0.7f);  // Instantiate the
    RNAGraphBMP class object
1541                g_pRNAGraphBMP->LoadFromRNASequence(g_pRNA->getTitle(), g_pRNA->
    getSequence(), g_pRNA->getStepPosition(), g_pRNA->getPairing());
1542
1543                delete g_pCircular;
1544                g_pCircular = new CircularGraph;  // Instantiate the CircularGraph
    class object
1545                delete g_pMatrix;
1546                g_pMatrix = new MatrixGraph;  // Instantiate the MatrixGraph class
    object
1547                delete g_pProgress;
1548                g_pProgress = new ProgressGraph; // Instantiate the ProgressGraph
    class object
1549                delete g_pBracketed;
1550                g_pBracketed = new BracketedGraph; // Instantiate the BracketedGraph
    class object
1551
1552                CreateDynamicTexture();
1553                g_pStage->SetTextureFromBitmap(g_hbm);
1554                UpdateStageLines();
1555
1556                EndDialog(hDlg, LOWORD(wParam));
1557                return (INT_PTR)TRUE;
1558            }
1559            break;
1560        case IDCANCEL:
1561            EndDialog(hDlg, LOWORD(wParam));
1562            return (INT_PTR)TRUE;
1563            break;
1564        }
1565        break;
1566    }
1567    return (INT_PTR)FALSE;
1568 }
1569
1570 // checkFASTA: Checks to see if the sequence is a FASTA sequence and if so it removes
1571 // the FASTA text so that only the RNA sequence remains
1572 int checkFASTA(char* str) {
1573    int i = 0; // index to copy to
1574    int j = 0; // index to copy from
1575
1576    if(*str == '>') // This is a FASTA sequence
1577    {
1578        while(*(str + j) != '\n') { // Loop to end of first line
1579            j++;
1580        }
1581        j++;
1582        while(*(str + j) != '\0') { // Loop to end of sequence
1583            *(str + i++) = *(str + j++);
1584        }
1585        *(str + i) = '\0'; // Terminate the character array
1586    }
1587
1588    return i; // Return length of updated character array
1589 }
```

```
1590
1591 // checkRNA: Checks to make sure that only valid characters (ACGU) are in
1592 // the RNA sequence and removes everything else. Converts lower case to upper case
1593 int checkRNA(char* str) {
1594     int i = 0; // index to copy to
1595     int j = 0; // index to copy from
1596
1597     while(*(str + j) != '\0') { // Loop to end of sequence
1598         if(*(str + j) == 'A' || *(str + j) == 'C' || *(str + j) == 'G' || *(str + j) ↙
      == 'U')
1599             *(str + i++) = *(str + j++);
1600         else
1601             if(*(str + j) == 'a' || *(str + j) == 'c' || *(str + j) == 'g' || *(str +↙
       j) == 'u')
1602                 *(str + i++) = *(str + j++) - 32; // Convert to upper case
1603             else
1604                 j++;
1605     }
1606     *(str + i) = '\0'; // Terminate the character array
1607
1608     return i; // Return length of updated character array
1609 }
1610
1611 // FUNCTION:  UpdateStageLines
1612 // Used to update the lines that will be drawn between the planar graph and the      ↙
      matrix grap
1613 //
1614 // @param   undefined   void
1615 // @return  void
1616 void UpdateStageLines()
1617 {
1618     LineList listLines;  // Instantiate the list which holds lines
1619     if(g_DrawLines)
1620     {
1621         List::Node_type *node_ptr = g_pNussinov->listTraceback.head;
1622         while(node_ptr != NULL)
1623         {
1624             if(node_ptr->next != NULL)
1625             {
1626                 if(node_ptr->next->i == node_ptr->i + 1 && node_ptr->next->j ==      ↙
      node_ptr->j)
1627                 {   // i Unpaired
1628                     // Draw a line from current traceback cell to base at position i
1629                     listLines.Add(node_ptr->xPos, node_ptr->yPos, g_pRNA->           ↙
      GetPlanarPos()[node_ptr->i].PlanarX, g_pRNA->GetPlanarPos()[node_ptr->i].PlanarY,↙
       D3DCOLOR_RGBA(102, 9, 123, 255));
1630                 }
1631                 else
1632                 {
1633                     if(node_ptr->next->i == node_ptr->i && node_ptr->next->j ==      ↙
      node_ptr->j - 1)
1634                     {   // j Unpaired
1635                         // Draw a line from current traceback cell to base at       ↙
      position j
1636                         listLines.Add(node_ptr->xPos, node_ptr->yPos, g_pRNA->       ↙
      GetPlanarPos()[node_ptr->j].PlanarX, g_pRNA->GetPlanarPos()[node_ptr->j].PlanarY,↙
       D3DCOLOR_RGBA(223, 0, 41, 255));
1637                     }
1638                     else
1639                     {
1640                         if(node_ptr->next->i == node_ptr->i + 1 && node_ptr->next->j ↙
      == node_ptr->j - 1)
1641                         {   // i,j Paired
1642                             // Draw a line from current traceback cell to both base ↙
      at position i and position j
1643                             listLines.Add(node_ptr->xPos, node_ptr->yPos, g_pRNA->   ↙
      GetPlanarPos()[node_ptr->i].PlanarX, g_pRNA->GetPlanarPos()[node_ptr->i].PlanarY,↙
```

```
     D3DCOLOR_RGBA(0, 105, 179, 255));
1644                         listLines.Add(node_ptr->xPos, node_ptr->yPos, g_pRNA->
    GetPlanarPos()[node_ptr->j].PlanarX, g_pRNA->GetPlanarPos()[node_ptr->j].PlanarY,
     D3DCOLOR_RGBA(0, 105, 179, 255));
1645                     }
1646                     else
1647                     {   // Bifurcation
1648                         // Draw a line from current traceback cell to both base
    at position i and position j
1649                         listLines.Add(node_ptr->xPos, node_ptr->yPos, g_pRNA->
    GetPlanarPos()[node_ptr->i].PlanarX, g_pRNA->GetPlanarPos()[node_ptr->i].PlanarY,
     D3DCOLOR_RGBA(0, 159, 98, 255));
1650                         listLines.Add(node_ptr->xPos, node_ptr->yPos, g_pRNA->
    GetPlanarPos()[node_ptr->j].PlanarX, g_pRNA->GetPlanarPos()[node_ptr->j].PlanarY,
     D3DCOLOR_RGBA(0, 159, 98, 255));
1651                     }
1652                 }
1653             }
1654         }
1655         else
1656         {
1657             listLines.Add(node_ptr->xPos, node_ptr->yPos, g_pRNA->GetPlanarPos()
    [node_ptr->i].PlanarX, g_pRNA->GetPlanarPos()[node_ptr->i].PlanarY, D3DCOLOR_RGBA
    (0, 0, 0, 255));
1658         }
1659         node_ptr = node_ptr->next;
1660     }
1661 }
1662 g_pStage->CreateLines(&listLines);
1663 listLines.Clear();
1664 }
1665
1666 // FUNCTION:  GetCurrentPath
1667 // Used to obtain the currrent working directory
1668 //
1669 // @param   buffer   char*   the buffer to store the working directory into
1670 // @return  void
1671 void GetCurrentPath(char* buffer)
1672 {
1673     getcwd(buffer, 150);
1674 }
1675
```

```
1  //-----------------------------------------------------------------------------
2  // Copyright (c): 2006, All Rights Reserved
3  // Project:       SJSU Masters Project
4  // File:          Stage.h
5  // Purpose:       Header file for Stage class.  This class is the main object which
6  //                the facets will be drawn on.
7  //
8  // Start Date:    2/1/2006
9  // Programmer:    Brandon Hunter
10 //-----------------------------------------------------------------------------
11
12 #pragma once
13
14 #include <d3dx9.h>
15 #include "LineList.h" // Header file for linked list of lines
16
17 //Define a FVF for the Stage
18 #define STAGE_D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_TEX1)
19 //Define a FVF for the lines
20 //#define LINE_D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ | D3DFVF_DIFFUSE)
21 #define LINE_D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_DIFFUSE)
22
23 class Stage
24 {
25 private:
26     //Define a custom vertex for our stage
27     struct STAGE_CUSTOMVERTEX
28     {
29         float x, y, z;      //Position of vertex in 3D space
30         float nx, ny, nz;   //Lighting Normal
31         float tu, tv;       //Texture coordinates
32     };
33
34     //struct LINE_CUSTOMVERTEX
35     //{
36     //  float x, y, z;      // Position of vertex in 3D space
37     //  DWORD color;        // Color of vertex
38     //};
39     struct LINE_CUSTOMVERTEX
40     {
41         float x, y, z;      // Position of vertex in 3D space
42         float nx, ny, nz;   // Normal vector for lighting calculations
43         DWORD color;        // Diffuse color of vertex
44     };
45
46
47 public:
48     float m_xPos;
49     float m_yPos;
50     float m_zPos;
51     float m_fTheta; // This is the angle of the wings on the stage
52     float m_fScale;
53     float m_fPitch;
54     float m_fYaw;
55     float m_fRoll;
56     bool SetMaterial(D3DCOLORVALUE rgbaDiffuse, D3DCOLORVALUE rgbaAmbient,
       D3DCOLORVALUE rgbaSpecular, D3DCOLORVALUE rgbaEmissive, float rPower);
57     bool SetTexture(const char* szTextureFilePath);
58     bool SetTextureFromBitmap(HBITMAP hBitmap);
59     bool SetPosition(float x, float y, float z);
60     bool SetStageAngle(float Theta);
61     DWORD Render();
62     Stage(LPDIRECT3DDEVICE9 pD3DDevice, float Theta = D3DXToRadian(25.0f), float x = 0.
       0f, float y = 0.0f, float z = 0.0f);
63     ~Stage();
```

```
64      void CreateLines(LineList* plistLines);
65
66  private:
67      bool CreateIndexBuffer();
68      D3DXVECTOR3 GetTriangeNormal(D3DXVECTOR3* vVertex1, D3DXVECTOR3* vVertex2,
        D3DXVECTOR3* vVertex3);
69      bool UpdateVertices();
70      HRESULT CreateVertexBuffer();
71      LPDIRECT3DDEVICE9 m_pD3DDevice;
72      LPDIRECT3DVERTEXBUFFER9 m_pVertexBuffer;
73      LPDIRECT3DTEXTURE9 m_pTexture;
74      D3DMATERIAL9 m_matMaterial;
75      LPDIRECT3DINDEXBUFFER9 m_pIndexBuffer;
76      LPDIRECT3DVERTEXBUFFER9 m_pLineVertexBuffer;
77
78      DWORD m_dwNumOfVertices;
79      DWORD m_dwNumOfIndices;
80      DWORD m_dwNumOfPolygons;
81      int m_lineCount;
82  };
```

```
1  //-----------------------------------------------------------------------------
2  // Copyright (c): 2006, All Rights Reserved
3  // Project:       SJSU Masters Project
4  // File:          Stage.h
5  // Purpose:       This is the class implementation of the Stage object.  This class is
6  //                the main object which the facets will be drawn on.
7  //
8  // Start Date:    2/1/2006
9  // Programmer:    Brandon Hunter
10 //-----------------------------------------------------------------------------
11
12 #include "Stage.h"
13
14 // FUNCTION:  Stage
15 // Default Constructor
16 //
17 // @param   pD3DDevice   LPDIRECT3DDEVICE9   the Direct3D device
18 // @param   Theta        float               the angle that the stage is open at
19 // @param   x            float               x coordinate position
20 // @param   y            float               y coordinate position
21 // @param   z            float               x coordinate position
22 // @return  void
23 Stage::Stage(LPDIRECT3DDEVICE9 pD3DDevice, float Theta, float x, float y, float z)
24 {
25     m_pD3DDevice = pD3DDevice;
26     m_pVertexBuffer = NULL;
27     m_pIndexBuffer = NULL;
28     m_pTexture = NULL;
29     m_pLineVertexBuffer = NULL;
30
31     //Setup counts for this object
32     m_dwNumOfVertices = 50;
33     m_dwNumOfIndices  = 72;
34     m_dwNumOfPolygons = 24;
35     m_lineCount = 0;
36
37     //Set a default size and position
38     m_fTheta = D3DXToRadian(Theta);
39     m_xPos = x;
40     m_yPos = y;
41     m_zPos = z;
42     m_fScale = 1.0f;
43     m_fPitch = 0.0f;
44     m_fYaw = 0.0f;
45     m_fRoll = 0.0f;
46
47     //Set material default values (R, G, B, A)
48     D3DCOLORVALUE rgbaDiffuse  = {1.0, 1.0, 1.0, 0.0,};
49     D3DCOLORVALUE rgbaAmbient  = {1.0, 1.0, 1.0, 0.0,};
50     D3DCOLORVALUE rgbaSpecular = {0.0, 0.0, 0.0, 0.0,};
51     D3DCOLORVALUE rgbaEmissive = {0.0, 0.0, 0.0, 0.0,};
52
53     SetMaterial(rgbaDiffuse, rgbaAmbient, rgbaSpecular, rgbaEmissive, 0);
54
55     //Initialize Vertex Buffer
56     if(SUCCEEDED(CreateVertexBuffer()))
57     {
58         if(CreateIndexBuffer())
59         {
60             UpdateVertices();
61         }
62     }
63 }
64
```

```
65 // FUNCTION:  ~Stage
66 // Default Destructor
67 //
68 // @param   undefined   void
69 // @return  void
70 Stage::~Stage()
71 {
72     if(m_pTexture != NULL)
73     {
74         m_pTexture->Release();
75         m_pTexture = NULL;
76     }
77     if(m_pIndexBuffer != NULL)
78     {
79         m_pIndexBuffer->Release();
80         m_pIndexBuffer = NULL;
81     }
82     if(m_pVertexBuffer != NULL)
83     {
84         m_pVertexBuffer->Release();
85         m_pVertexBuffer = NULL;
86     }
87     if(m_pLineVertexBuffer != NULL)
88     {
89         m_pLineVertexBuffer->Release();
90         m_pLineVertexBuffer = NULL;
91     }
92 }
93
94 // FUNCTION:  Render
95 // Used to render the stage to the screen
96 //
97 // @param   undefined   void
98 // @return  DWORD
99 DWORD Stage::Render()
100 {
101     m_pD3DDevice->SetStreamSource(0, m_pVertexBuffer, 0, sizeof(struct
      STAGE_CUSTOMVERTEX));
102     //m_pD3DDevice->SetVertexShader(STAGE_D3DFVF_CUSTOMVERTEX);
103     m_pD3DDevice->SetFVF(STAGE_D3DFVF_CUSTOMVERTEX);
104
105     if(m_pTexture != NULL)
106     {
107         //A texture has been set. We want our texture to be shaded based
108         //on the current light levels, so use D3DTOP_MODULATE.
109         m_pD3DDevice->SetTexture(0, m_pTexture);
110         m_pD3DDevice->SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_MODULATE);
111         m_pD3DDevice->SetTextureStageState(0, D3DTSS_COLORARG1, D3DTA_TEXTURE);
112         m_pD3DDevice->SetTextureStageState(0, D3DTSS_COLORARG2, D3DTA_CURRENT);
113     }
114     else
115     {
116         //No texture has been set
117         m_pD3DDevice->SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_SELECTARG2);
118         m_pD3DDevice->SetTextureStageState(0, D3DTSS_COLORARG1, D3DTA_TEXTURE);
119         m_pD3DDevice->SetTextureStageState(0, D3DTSS_COLORARG2, D3DTA_CURRENT);
120     }
121
122     m_pD3DDevice->SetMaterial(&m_matMaterial); //Select the material to use
123     m_pD3DDevice->SetIndices(m_pIndexBuffer); //Select index buffer
124     m_pD3DDevice->DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 0, m_dwNumOfVertices, 0,
       m_dwNumOfPolygons); //Render polygons from index buffer
125
126     // Render lines
127     if(m_lineCount > 0)
128     {
129         m_pD3DDevice->SetStreamSource(0, m_pLineVertexBuffer, 0, sizeof(struct
```

```
    LINE_CUSTOMVERTEX));
130         m_pD3DDevice->SetFVF(LINE_D3DFVF_CUSTOMVERTEX );
131         m_pD3DDevice->DrawPrimitive(D3DPT_LINELIST, 0, m_lineCount);
132     }
133 
134     //Return the number of polygons rendered
135     return m_dwNumOfPolygons;
136 }
137 
138 // FUNCTION:  CreateLines
139 // Used to create the lines drawn between the planar graph and matrix graph
140 //
141 // @param   plistLines   LineList*   pointer to list of lines
142 // @return  void
143 void Stage::CreateLines(LineList* plistLines)
144 {
145     VOID* pVertices;
146 
147     m_lineCount = 0;
148 
149     //Initialize Line Vertex Buffer
150     if(m_pLineVertexBuffer != NULL)
151     {
152         m_pLineVertexBuffer->Release();
153         m_pLineVertexBuffer = NULL;
154     }
155 
156     //Create the Line vertex buffer from our device.
157     if(SUCCEEDED(m_pD3DDevice->CreateVertexBuffer((plistLines->count * 2) * sizeof
    (struct LINE_CUSTOMVERTEX),
158                                                   0, LINE_D3DFVF_CUSTOMVERTEX,
159                                                   D3DPOOL_DEFAULT, &m_pLineVertexBuffer,
    NULL)))
160     {
161         LINE_CUSTOMVERTEX* p_cvLineVertices = new LINE_CUSTOMVERTEX[plistLines->count
    * 2];
162 
163         float scaleX = 10.0f / 512.0f;
164         float scaleY = 10.0f / 512.0f;
165         int x = 0;
166         LineList::Node_type* node_ptr = plistLines->head;
167         while(node_ptr != NULL)
168         {
169             p_cvLineVertices[x].x = (node_ptr->startX  * scaleX) - 5.0f;
170             p_cvLineVertices[x].y = 0.0f;
171             p_cvLineVertices[x].z = 12.5f - (node_ptr->startY * scaleY);
172             p_cvLineVertices[x].nx = 0.0f;
173             p_cvLineVertices[x].ny = 1.0f;
174             p_cvLineVertices[x].nz = 0.0f;
175             p_cvLineVertices[x].color = node_ptr->color;
176             x++;
177             p_cvLineVertices[x].x = (node_ptr->endX  * scaleX) - 5.0f;
178             p_cvLineVertices[x].y = 10.0f - (node_ptr->endY * scaleY);
179             p_cvLineVertices[x].z = 12.5;
180             p_cvLineVertices[x].nx = 0.0f;
181             p_cvLineVertices[x].ny = 1.0f;
182             p_cvLineVertices[x].nz = 0.0f;
183             p_cvLineVertices[x].color = node_ptr->color;
184             x++;
185             node_ptr = node_ptr->next;
186         }
187 
188         //Get a pointer to the Line vertex buffer vertices and lock the vertex buffer
189         if(SUCCEEDED(m_pLineVertexBuffer->Lock(0, sizeof(LINE_CUSTOMVERTEX) *
    (plistLines->count * 2), (void**)&pVertices, 0)))
190         {
191             //Copy our stored vertices values into the vertex buffer
```

```
192             memcpy(pVertices, p_cvLineVertices, sizeof(LINE_CUSTOMVERTEX) *
    (plistLines->count * 2));
193             //Unlock the vertex buffer
194             m_pLineVertexBuffer->Unlock();
195
196             m_lineCount = plistLines->count;
197         }
198
199         delete[] p_cvLineVertices; // delete dynamically allocated memory
200     }
201 }
202
203 // FUNCTION:  CreateVertexBuffer
204 // Used to create the vertex buffer
205 //
206 // @param   undefined   void
207 // @return  HRESULT
208 HRESULT Stage::CreateVertexBuffer()
209 {
210     //Create the vertex buffer from our device.
211     if(FAILED(m_pD3DDevice->CreateVertexBuffer(m_dwNumOfVertices * sizeof(struct
    STAGE_CUSTOMVERTEX),
212                                                0, STAGE_D3DFVF_CUSTOMVERTEX,
213                                                D3DPOOL_DEFAULT, &m_pVertexBuffer,
    NULL)))
214     {
215         return E_FAIL;
216     }
217     return S_OK;
218 }
219
220 // FUNCTION:  CreateIndexBuffer
221 // Used to create the index buffer
222 //
223 // @param   undefined   void
224 // @return  bool
225 bool Stage::CreateIndexBuffer()
226 {
227     VOID* pBufferIndices;
228
229     //Create the index buffer from our device
230     if(FAILED(m_pD3DDevice->CreateIndexBuffer(m_dwNumOfIndices * sizeof(WORD),
231                                               0, D3DFMT_INDEX16, D3DPOOL_MANAGED,
232                                               &m_pIndexBuffer, NULL)))
233     {
234         return false;
235     }
236
237     //Set values for the index buffer
238     WORD pIndices[] = { 0, 1, 2, 3, 2, 1,   //Face 0
239                         4, 5, 6, 7, 6, 5,   //Face 1
240                         8, 9,10,11,10, 9,   //Face 2
241                        12,13,14,15,14,13,   //Face 3
242                        16,17,18,19,18,17,   //Face 4
243                        20,21,22,23,22,21,   //Face 5
244                        24,25,26,27,26,25,   //Face 6
245                        28,29,30,            //Face 7
246                        31,32,33,34,33,32,   //Face 8
247                        35,36,37,            //Face 9
248                        38,39,40,41,40,39,   //Face 10
249                        42,43,44,45,44,43,   //Face 11
250                        46,47,48,49,48,47};  //Face 12
251
252     //WORD pIndices[] = { 0, 1, 2, 3, 2, 1, //Top
253     //                    4, 5, 6, 7, 6, 5,   //Face 1
254     //                    8, 9,10,11,10, 9,   //Face 2
255     //                   12,13,14,15,14,13,   //Face 3
```

```
256     //                  16,17,18,19,18,17,   //Face 4
257     //                  20,21,22,23,22,21};  //Bottom
258
259     //Get a pointer to the index buffer indices and lock the index buffer
260     m_pIndexBuffer->Lock(0, m_dwNumOfIndices * sizeof(WORD), (void**)&pBufferIndices,
        0);
261
262     //Copy our stored indices values into the index buffer
263     memcpy(pBufferIndices, pIndices, m_dwNumOfIndices * sizeof(WORD));
264
265     //Unlock the index buffer
266     m_pIndexBuffer->Unlock();
267
268     return true;
269 }
270
271 // FUNCTION:  GetTriangleNormal
272 // Calculate the triangle normals.  Used to lighting
273 //
274 // @param   vVertex1   D3DXVECTOR3*   First triangle vertex
275 // @param   vVertex2   D3DXVECTOR3*   Second triangle vertex
276 // @param   vVertex3   D3DXVECTOR3*   Third triangle vertex
277 // @return  D3DXVECTOR3
278 D3DXVECTOR3 Stage::GetTriangeNormal(D3DXVECTOR3* vVertex1, D3DXVECTOR3* vVertex2,
        D3DXVECTOR3* vVertex3)
279 {
280     D3DXVECTOR3 vNormal;
281     D3DXVECTOR3 v1;
282     D3DXVECTOR3 v2;
283
284     D3DXVec3Subtract(&v1, vVertex2, vVertex1);
285     D3DXVec3Subtract(&v2, vVertex3, vVertex1);
286
287     D3DXVec3Cross(&vNormal, &v1, &v2);
288
289     D3DXVec3Normalize(&vNormal, &vNormal);
290
291     return vNormal;
292 }
293
294 // FUNCTION:  UpdateVertices
295 // Used to update the vertices
296 //
297 // @param   undefined   void
298 // @return  bool
299 bool Stage::UpdateVertices()
300 {
301     DWORD i;
302     VOID* pVertices;
303     WORD* pBufferIndices;
304     D3DXVECTOR3 vNormal;
305     DWORD dwVertex1;
306     DWORD dwVertex2;
307     DWORD dwVertex3;
308
309     WORD* pNumOfSharedPolygons = new WORD[m_dwNumOfVertices];           //Array holds
        how many times this vertex is shared
310     D3DVECTOR* pSumVertexNormal = new D3DVECTOR[m_dwNumOfVertices];     //Array holds
        sum of all face normals for shared vertex
311
312     //Clear memory
313     for(i = 0; i < m_dwNumOfVertices; i++)
314     {
315         pNumOfSharedPolygons[i] = 0;
316         pSumVertexNormal[i] = D3DXVECTOR3(0,0,0);
317     }
318
```

```
319     STAGE_CUSTOMVERTEX cvVertices[] =
320     {
321         // Face 0
322         {-5 - (10 * sin(m_fTheta)),  -2,   0, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,},       //↙
    Vertex 0
323         {-5 - (10 * sin(m_fTheta)),   0,   0, 0.0f, 0.0f, 0.0f, 0.0f, 0.9f,},       //↙
    Vertex 1
324         {5 + (10 * sin(m_fTheta)),  -2,   0, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f,},        //↙
    Vertex 2
325         {5 + (10 * sin(m_fTheta)),   0,   0, 0.0f, 0.0f, 0.0f, 1.0f, 0.9f,},        //↙
    Vertex 3
326
327         // Face 1
328         { -5,   0,   0, 0.0f, 0.0f, 0.0f, 0.0f, 0.9f,},     //Vertex 4
329         { -5,   0, 2.5, 0.0f, 0.0f, 0.0f, 0.0f, 0.8f,},     //Vertex 5
330         {-2.5,  0,   0, 0.0f, 0.0f, 0.0f, 0.125f, 0.9f,},       //Vertex 6
331         {-2.5,  0, 2.5, 0.0f, 0.0f, 0.0f, 0.125f, 0.8f,},       //Vertex 7
332
333         // Face 2
334         {-2.5,  0,   0, 0.0f, 0.0f, 0.0f, 0.125f, 0.9f,},       //Vertex 8
335         {-2.5,  0, 2.5, 0.0f, 0.0f, 0.0f, 0.125f, 0.8f,},       //Vertex 9
336         {  0,   0,   0, 0.0f, 0.0f, 0.0f, 0.25f, 0.9f,},        //Vertex 10
337         {  0,   0, 2.5, 0.0f, 0.0f, 0.0f, 0.250f, 0.8f,},       //Vertex 11
338
339         // Face 3
340         {  0,   0,   0, 0.0f, 0.0f, 0.0f, 0.25f, 0.9f,},        //Vertex 12
341         {  0,   0, 2.5, 0.0f, 0.0f, 0.0f, 0.25f, 0.8f,},        //Vertex 13
342         {2.5,   0,   0, 0.0f, 0.0f, 0.0f, 0.375f, 0.9f,},       //Vertex 14
343         {2.5,   0, 2.5, 0.0f, 0.0f, 0.0f, 0.375f, 0.8f,},       //Vertex 15
344
345         // Face 4
346         {2.5,   0,   0, 0.0f, 0.0f, 0.0f, 0.375f, 0.9f,},       //Vertex 16
347         {2.5,   0, 2.5, 0.0f, 0.0f, 0.0f, 0.375f, 0.8f,},       //Vertex 17
348         {  5,   0,   0, 0.0f, 0.0f, 0.0f, 0.5f, 0.9f,},     //Vertex 18
349         {  5,   0, 2.5, 0.0f, 0.0f, 0.0f, 0.5f, 0.8f,},     //Vertex 19
350
351         // Face 5
352         {-5 - (10 * sin(m_fTheta)),   0,   0, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,},       //↙
    Vertex 20
353         {-5 - (10 * sin(m_fTheta)),   0, 12.5 - (10 * cos(m_fTheta)), 0.0f, 0.0f, 0.0f↙
    , 0.0f, 0.0f,},     //Vertex 21
354         { -5,   0,   0, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,},     //Vertex 22
355         { -5,   0, 12.5 - (10 * cos(m_fTheta)), 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,},     //↙
    Vertex 23
356
357         // Face 6
358         {  5,   0,   0, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,},     //Vertex 24
359         {  5,   0, 12.5 - (10 * cos(m_fTheta)), 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,},     //↙
    Vertex 25
360         {5 + (10 * sin(m_fTheta)),   0,   0, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,},        //↙
    Vertex 26
361         {5 + (10 * sin(m_fTheta)),   0, 12.5 - (10 * cos(m_fTheta)), 0.0f, 0.0f, 0.0f,↙
     0.0f, 0.0f,},      //Vertex 27
362
363         // Face 7
364         {-5 - (10 * sin(m_fTheta)),   0, 12.5 - (10 * cos(m_fTheta)), 0.0f, 0.0f, 0.0f↙
    , 0.0f, 0.0f,},     //Vertex 28
365         { -5,   0,12.5, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,},     //Vertex 29
366         { -5,   0, 12.5 - (10 * cos(m_fTheta)), 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,},     //↙
    Vertex 30
367
368         // Face 8
369         { -5,   0, 2.5, 0.0f, 0.0f, 0.0f, 0.0f, 0.8f,},     //Vertex 31
370         { -5,   0,12.5, 0.0f, 0.0f, 0.0f, 0.0f, 0.4f,},     //Vertex 32
371         {  5,   0, 2.5, 0.0f, 0.0f, 0.0f, 0.5f, 0.8f,},     //Vertex 33
372         {  5,   0,12.5, 0.0f, 0.0f, 0.0f, 0.5f, 0.4f,},     //Vertex 34
373
```

```
374         // Face 9
375         {  5,   0, 12.5 - (10 * cos(m_fTheta)), 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,},     //Vertex 35
376         {  5,   0,12.5, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,},     //Vertex 36
377         {5 + (10 * sin(m_fTheta)),   0, 12.5 - (10 * cos(m_fTheta)), 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,},      //Vertex 37
378
379         // Face 10
380         {-5 - (10 * sin(m_fTheta)),   0, 12.5 - (10 * cos(m_fTheta)), 0.0f, 0.0f, 0.0f, 0.5f, 0.8f,},     //Vertex 38
381         {-5 - (10 * sin(m_fTheta)),  10, 12.5 - (10 * cos(m_fTheta)), 0.0f, 0.0f, 0.0f, 0.5f, 0.4f,},     //Vertex 39
382         { -5,   0,12.5, 0.0f, 0.0f, 0.0f, 1.0f, 0.8f,},     //Vertex 40
383         { -5,  10,12.5, 0.0f, 0.0f, 0.0f, 1.0f, 0.4f,},     //Vertex 41
384
385         // Face 11
386         { -5,   0,12.5, 0.0f, 0.0f, 0.0f, 0.0f, 0.4f,},     //Vertex 42
387         { -5,  10,12.5, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,},     //Vertex 43
388         {  5,   0,12.5, 0.0f, 0.0f, 0.0f, 0.5f, 0.4f,},     //Vertex 44
389         {  5,  10,12.5, 0.0f, 0.0f, 0.0f, 0.5f, 0.0f,},     //Vertex 45
390
391         // Face 12
392         {  5,   0,12.5, 0.0f, 0.0f, 0.0f, 0.5f, 0.4f,},     //Vertex 46
393         {  5,  10,12.5, 0.0f, 0.0f, 0.0f, 0.5f, 0.0f,},     //Vertex 47
394         {5 + (10 * sin(m_fTheta)),   0, 12.5 - (10 * cos(m_fTheta)), 0.0f, 0.0f, 0.0f, 1.0f, 0.4f,},      //Vertex 48
395         {5 + (10 * sin(m_fTheta)),  10, 12.5 - (10 * cos(m_fTheta)), 0.0f, 0.0f, 0.0f, 1.0f, 0.0f,},      //Vertex 49
396     };
397
398     //Get a pointer to the index buffer indices and lock the index buffer
399     m_pIndexBuffer->Lock(0, m_dwNumOfIndices * sizeof(WORD), (void**)&pBufferIndices, D3DLOCK_READONLY);
400
401     //For each triangle, count the number of times each vertex is used and
402     //add together the normals of faces that share a vertex
403     for(i = 0; i < m_dwNumOfIndices; i += 3)
404     {
405         dwVertex1 = pBufferIndices[i];
406         dwVertex2 = pBufferIndices[i + 1];
407         dwVertex3 = pBufferIndices[i + 2];
408
409
410         vNormal = GetTriangeNormal(&D3DXVECTOR3(cvVertices[dwVertex1].x, cvVertices[dwVertex1].y, cvVertices[dwVertex1].z),
411                                    &D3DXVECTOR3(cvVertices[dwVertex2].x, cvVertices[dwVertex2].y, cvVertices[dwVertex2].z),
412                                    &D3DXVECTOR3(cvVertices[dwVertex3].x, cvVertices[dwVertex3].y, cvVertices[dwVertex3].z));
413
414
415         pNumOfSharedPolygons[dwVertex1]++;
416         pNumOfSharedPolygons[dwVertex2]++;
417         pNumOfSharedPolygons[dwVertex3]++;
418
419         pSumVertexNormal[dwVertex1].x += vNormal.x;
420         pSumVertexNormal[dwVertex1].y += vNormal.y;
421         pSumVertexNormal[dwVertex1].z += vNormal.z;
422
423         pSumVertexNormal[dwVertex2].x += vNormal.x;
424         pSumVertexNormal[dwVertex2].y += vNormal.y;
425         pSumVertexNormal[dwVertex2].z += vNormal.z;
426
427         pSumVertexNormal[dwVertex3].x += vNormal.x;
428         pSumVertexNormal[dwVertex3].y += vNormal.y;
429         pSumVertexNormal[dwVertex3].z += vNormal.z;
430     }
```

```
431
432     //Unlock the index buffer
433     m_pIndexBuffer->Unlock();
434
435
436     //For each vertex, calculate and set the average normal
437     for(i = 0; i < m_dwNumOfVertices; i++)
438     {
439         vNormal.x = pSumVertexNormal[i].x / pNumOfSharedPolygons[i];
440         vNormal.y = pSumVertexNormal[i].y / pNumOfSharedPolygons[i];
441         vNormal.z = pSumVertexNormal[i].z / pNumOfSharedPolygons[i];
442
443         D3DXVec3Normalize(&vNormal, &vNormal);
444
445         cvVertices[i].nx = vNormal.x;
446         cvVertices[i].ny = vNormal.y;
447         cvVertices[i].nz = vNormal.z;
448     }
449
450
451     //Get a pointer to the vertex buffer vertices and lock the vertex buffer
452     if(FAILED(m_pVertexBuffer->Lock(0, sizeof(cvVertices), (void**)&pVertices, 0)))
453     {
454         return false;
455     }
456
457     //Copy our stored vertices values into the vertex buffer
458     memcpy(pVertices, cvVertices, sizeof(cvVertices));
459
460     //Unlock the vertex buffer
461     m_pVertexBuffer->Unlock();
462
463     //Clean up
464     delete pNumOfSharedPolygons;
465     delete pSumVertexNormal;
466
467     pNumOfSharedPolygons = NULL;
468     pSumVertexNormal = NULL;
469
470     return true;
471 }
472
473 // FUNCTION:  SetStageAngle
474 // Used to set the stage angle
475 //
476 // @param   Theta   float   the angle to set the stage
477 // @return  bool
478 bool Stage::SetStageAngle(float Theta)
479 {
480     m_fTheta = Theta;
481
482     UpdateVertices();
483
484     return true;
485 }
486
487 // FUNCTION:  SetPosition
488 // Used to set the stage position
489 //
490 // @param   x    float    the x coordinate position
491 // @param   y    float    the y coordinate position
492 // @param   z    float    the z coordinate position
493 // @return  bool
494 bool Stage::SetPosition(float x, float y, float z)
495 {
496     m_xPos = x;
497     m_yPos = y;
```

```
498     m_zPos = z;
499
500     UpdateVertices();
501
502     return true;
503 }
504
505 // FUNCTION:  SetTexture
506 // Used to set the texture from a disk file
507 //
508 // @param   szTextureFilePath  const char*   the path to the texture file
509 // @return  bool
510 bool Stage::SetTexture(const char *szTextureFilePath)
511 {
512     if(FAILED(D3DXCreateTextureFromFileA(m_pD3DDevice, szTextureFilePath, &
        m_pTexture)))
513     {
514         return false;
515     }
516
517     return true;
518 }
519
520 // FUNCTION:  SetTextureFromBitmap
521 // Used to create the texture from a bitmap in memory
522 //
523 // @param   hBitmap   HBITMAP   handle to a bitmap
524 // @return  bool
525 bool Stage::SetTextureFromBitmap(HBITMAP hBitmap)
526 {
527     if(m_pTexture == NULL)
528     {
529         D3DXCreateTexture(m_pD3DDevice, 1024, 1280, 1, 0, D3DFMT_A8R8G8B8,
        D3DPOOL_MANAGED, &m_pTexture);
530         //D3DXCreateTexture(m_pD3DDevice, 1024, 1280, 1, D3DUSAGE_AUTOGENMIPMAP,
        D3DFMT_A8R8G8B8, D3DPOOL_MANAGED, &m_pTexture);
531     }
532     if(m_pTexture == NULL)
533         return m_pTexture;
534     if(hBitmap == NULL)
535         return hBitmap;
536
537     D3DSURFACE_DESC d3dsd;
538     m_pTexture->GetLevelDesc(0, &d3dsd);
539
540     SIZE size = {d3dsd.Width, d3dsd.Height};
541
542     D3DLOCKED_RECT rcLockedRect = { 0 };
543     RECT rc = { 0, 0, size.cx, size.cy };
544
545     m_pTexture->LockRect(0, &rcLockedRect, &rc, D3DLOCK_DISCARD);
546
547     BITMAP bmpX;
548     ZeroMemory(&bmpX, sizeof(bmpX));
549     GetObject(hBitmap, sizeof(bmpX), &bmpX);
550
551     BYTE* pTextureBits = (BYTE*)rcLockedRect.pBits;
552     GetBitmapBits(hBitmap, bmpX.bmWidthBytes * bmpX.bmHeight, pTextureBits);
553     m_pTexture->UnlockRect(0);
554
555     //BYTE* pTextureBits = (BYTE*)rcLockedRect.pBits;
556
557     //DWORD* pBitmapBits;
558     //DWORD* pBitmapBits = (DWORD*)bmp_info.bmBits;
559     //DWORD* pBitmapBits = (DWORD*)GlobalAlloc(GPTR, bmpX.bmWidthBytes * bmpX.
        bmHeight); //allocate memory for image byte buffer
560     //DWORD* pBitmapBits = new DWORD[bmpX.bmWidthBytes * bmpX.bmHeight]; // Create
```

```
    Memory for Bitmap Bits
561     //DWORD* pBitmapBits = (DWORD*)malloc(sizeof(DWORD) * (bmpX.bmWidthBytes * bmpX.bmHeight));
562     //GetBitmapBits(hBitmap, bmpX.bmWidthBytes * bmpX.bmHeight, pBitmapBits);
563
564     //BYTE* pLineTextureBits = pTextureBits;
565
566     //for (int j = 0; j < size.cy; j++)
567     //{
568     //    DWORD* pPixels = (DWORD*)pLineTextureBits;
569
570     //    for (int i = 0; i < size.cx; i++)
571     //    {
572     //        *pPixels = *pBitmapBits;
573
574     //        pPixels++;
575     //        pBitmapBits++;
576     //    }
577
578     //    pLineTextureBits += rcLockedRect.Pitch;
579     //}
580     //delete pBitmapBits; // Free Dynamic Memory
581     //free(pBitmapBits);
582     //GlobalFree(pBitmapBits);
583     //m_pTexture->UnlockRect(0);
584
585     m_pD3DDevice->SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
586     m_pD3DDevice->SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
587     m_pD3DDevice->SetSamplerState(0, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);
588     //m_pD3DDevice->SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_ANISOTROPIC);
589     //m_pD3DDevice->SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_ANISOTROPIC);
590     //m_pD3DDevice->SetSamplerState(0, D3DSAMP_MIPFILTER, D3DTEXF_ANISOTROPIC);
591 }
592
593 // FUNCTION:  SetMaterial
594 // Used to set the material properties used in the visualization
595 //
596 // @param   rgbaDiffuse   D3DCOLORVALUE
597 // @param   rgbaAmbient   D3DCOLORVALUE
598 // @param   rgbaSpecular  D3DCOLORVALUE
599 // @param   rgbaEmissive  D3DCOLORVALUE
600 // @param   rPower        float
601 // @return  bool
602 bool Stage::SetMaterial(D3DCOLORVALUE rgbaDiffuse, D3DCOLORVALUE rgbaAmbient, D3DCOLORVALUE rgbaSpecular, D3DCOLORVALUE rgbaEmissive, float rPower)
603 {
604     //Set the RGBA for diffuse light reflected from this material.
605     m_matMaterial.Diffuse = rgbaDiffuse;
606
607     //Set the RGBA for ambient light reflected from this material.
608     m_matMaterial.Ambient = rgbaAmbient;
609
610     //Set the color and sharpness of specular highlights for the material.
611     m_matMaterial.Specular = rgbaSpecular;
612     m_matMaterial.Power = rPower;
613
614     //Set the RGBA for light emitted from this material.
615     m_matMaterial.Emissive = rgbaEmissive;
616
617     return true;
618 }
```

```
1  /*****************************************************
2  * Copyright (c): 2006, All Rights Reserved
3  * Project:       SJSU Masters Project
4  * File:          Stack.h
5  * Purpose:       Header file for class implementation of a standard stack
6  *
7  * Start Date:    8/1/2006
8  * Programmer:    Brandon Hunter
9  *
10 *****************************************************/
11
12 #pragma once
13
14 class CStack
15 {
16 public:
17     typedef struct item_type {
18         int i;
19         int j;
20     } Item_type;
21
22     bool IsEmpty();
23     void Push(Item_type item);
24     void Pop(Item_type *item);
25     CStack();
26     virtual ~CStack();
27
28 private:
29     typedef struct node_tag {
30         Item_type info;
31         struct node_tag *next;
32     } Node_type;
33
34     Node_type *top;
35     Node_type* MakeNode(Item_type item);
36     void PushNode(Node_type *node_ptr);
37     void PopNode(Node_type **node_ptr);
38 };
```

```
/*****************************************************
* Copyright (c): 2006, All Rights Reserved
* Project:       SJSU Masters Project
* File:          Stack.cpp
* Purpose:       Class implementation of a standard stack
*
* Start Date:    8/1/2006
* Programmer:    Brandon Hunter
*
*****************************************************/

#include "StdAfx.h"
#include "Stack.h"

// FUNCTION:  CStack
// Default Constructor
//
// @param   undefined   void
// @return  void
CStack::CStack()
{
    top = NULL;  // Initialize top of stack
}

// FUNCTION:  ~CStack
// Default Destructor
//
// @param   undefined   void
// @return  void
CStack::~CStack()
{
    while(top != NULL) {
        Node_type *node_ptr;
        node_ptr = top;
        top = node_ptr->next;
        delete node_ptr;
    }
}

// FUNCTION:  Push
// Used to make a new node with item and push it onto stack
//
// @param   item    Item_type   the item to add to the stack
// @return  void
void CStack::Push(Item_type item) {
    PushNode(MakeNode(item));
}

// FUNCTION:  MakeNode
// Used to make a new node and insert item
//
// @param   item   Item_type   theh item to add
// @return  Node_type*
CStack::Node_type* CStack::MakeNode(Item_type item) {
    Node_type *p;

    if((p = new Node_type) == NULL)
        throw "CStack: Out of Memory";
    else {
        p->info = item;
        p->next = NULL;
    }
    return p;
}

// FUNCTION:  PushNode
// Used to push node onto the linked stack
```

```
68 //
69 // @param   node_ptr   Node_type*   the node to push on stack
70 // @return  void
71 void CStack::PushNode(Node_type *node_ptr) {
72     if(node_ptr == NULL)
73         throw "CStack: Attempted to push a nonexistent node";
74     else {
75         node_ptr->next = top;
76         top = node_ptr;
77     }
78 }
79
80 // FUNCTION:  Pop
81 // Used to pop a node from the stack and return its item
82 //
83 // @param   item   Item_type*   the item thats poped from the stack
84 // @return  void
85 void CStack::Pop(Item_type *item) {
86     Node_type *node_ptr;
87
88     PopNode(&node_ptr);
89     *item = node_ptr->info;
90     delete node_ptr;
91 }
92
93 // FUNCTION:  PopNode
94 // Used to pop node from the linked stack
95 //
96 // @param   node_ptr   Node_type**   the node to pop
97 // @return  void
98 void CStack::PopNode(Node_type **node_ptr) {
99     if(top == NULL)
100        throw "CStack: Empty Stack";
101    else {
102        *node_ptr = top;
103        top = (*node_ptr)->next;
104    }
105 }
106
107 // FUNCTION:  IsEmpty
108 // Used to determine of the stack is empty
109 //
110 // @param   undefined   void
111 // @return  bool
112 bool CStack::IsEmpty()
113 {
114     return top == NULL;
115 }
```

```
/*****************************************************
* Copyright (c): 2006, All Rights Reserved
* Project:       SJSU Masters Project
* File:          RNAGraphBMP.cpp
* Purpose:       To draw the secondary structure of RNA
*                into a bitmap surface using windows GDI
*
* Start Date:    8/1/2006
* Programmer:    Brandon Hunter
*
*****************************************************
*
* This program is partialy based on a program called
* plt22ps.c by Darrin Stewart Feb 26, 1998
*
*/

#pragma once
#include "rnagraph.h"
#include "RNA.h"

class RNAGraphBMP :
    public RNAGraph
{
private:
    float m_line_length;
    float m_init_view_x; // Initial View Matrix, x dimension
    float m_init_view_y; // Initial View Matrix, y dimension
    COLORREF m_current_color;
    float m_font_height;
    float m_line_width;
    float m_origin_x, m_origin_y;
    float m_user_tran_x, m_user_tran_y;
    float m_view_x, m_view_y;
    float m_dpiY; // Dots Per Inch, y dimension
    float m_xoff, m_yoff;

    float CalcFontHeight(float char_size, float scale);
    void remove_quotes(char *string);
    void DrawLine(HDC hdc, float x1, float y1, float x2, float y2);
    void DrawCircle(HDC hdc, float radius, float x, float y);
    void DrawLabel(HDC hdc, float angle, float x, float y, int number);
    void DrawCenteredText(HDC hdc, float x, float y, char *string);
    void DrawBase(HDC hdc, float x, float y, char base);
    void DrawLoopNumber(HDC hdc, float x, float y, int number);
    void DrawTitle(HDC hdc, char *title, int BMPSizeX, int BMPSizeY);
    float user_to_screen_x(float x);
    float user_to_screen_y(float y);

public:
    RNAGraphBMP(float MinBaseSeparation);
    ~RNAGraphBMP(void);

    void Draw(HDC hdc, float BMPScale, float xsize, float ysize, bool mark_loops, bool ↙
    draw_bases, float csz, int label_rate, bool dot_pairs, int mosaicx, int mosaicy,   ↙
    float glob_rot, RNA *pRNA);
};
```

```
/*****************************************************
* Copyright (c): 2006, All Rights Reserved
* Project:       SJSU Masters Project
* File:          RNAGraphBMP.cpp
* Purpose:       To draw the secondary structure of RNA
*                into a bitmap surface using windows GDI
*
* Start Date:    8/1/2006
* Programmer:    Brandon Hunter
*
*****************************************************
*
* This program is partialy based on a program called
* plt22ps.c by Darrin Stewart Feb 26, 1998
*
*/

#include "StdAfx.h"
#include "RNAGraphBMP.h"
#include <math.h>

// FUNCTION:  RNAGraphBMP
// Default Constructor
//
// @param   MinBaseSeparation   float  the minimum base separation
// @return  void
RNAGraphBMP::RNAGraphBMP(float MinBaseSeparation) : RNAGraph(MinBaseSeparation)
{
    // Initialize Variables
    m_line_length = 1.1f;
    m_init_view_x = 1.0f; // Initial View Matrix, x dimension
    m_init_view_y = 1.0f; // Initial View Matrix, y dimension
    m_origin_x = 0.0; // Set the origin of the screen cooridinates
    m_origin_y = 0.0;
    m_user_tran_x = 0.0;
    m_user_tran_y = 0.0;
    m_line_width = 0.0;
}

// FUNCTION:  ~RNAGraphBMP
// Default Destructor
//
// @param   undefined   void
// @return  void
RNAGraphBMP::~RNAGraphBMP(void)
{
}

// FUNCTION:  Draw
// Used to draw the planr graph on the device context
//
// screen cooridinates=view matrix*(user coordinates-user_tran_vector)+screen_tran_vec
//
// @param   hdc        HDC     handle to the bitmap device context
// @param   BMPScale   float   the scale size of the bitmap
// @param   xsize      float   the x coordinate of the bitmap size
// @param   ysize      float   the y coordinate of the bitmap size
// @param   mark_loops bool    flag to determine if loop numbers should be drawn
// @param   draw_bases bool    float to determine if base symbols should be drawn
// @param   csz        float   character size
// @param   label_rate int     the rate at which to label the sequence
// @param   dot_pairs  bool    flag to determine if a dot or line is drawn between
    bases
// @param   mosaicx    int     the number of frames in x dimension
// @param   mosaicy    int     the number of frames in y dimension
// @param   glob_rot   float   the global rotation factor
// @param   pRNA       RNA*    pointer to an RNA class
```

```
67 // @return  void
68 void RNAGraphBMP::Draw(HDC hdc, float BMPScale, float xsize, float ysize, bool
       mark_loops, bool draw_bases, float csz, int label_rate, bool dot_pairs, int
       mosaicx, int mosaicy, float glob_rot, RNA *pRNA) {
69     int i, mate, imx, imy, istart;
70     float xmin, xmax, ymin, ymax, scalex, scaley, scale, scalecsz;
71     float xmx, ymx;
72     float xn, yn, r, x1, y1, x2, y2, xs, ys, ct, st, xr, yr;
73     Loop* lp;
74
75     #define okx(x) (m_user_tran_x - scale <= (x) && (x) <= xmx + scale)
76     #define oky(y) (m_user_tran_y - scale <= (y) && (y) <= ymx + scale)
77
78     if(label_rate < 0) // Make sure label_rate is valid
79         label_rate = 0;
80
81     if(mosaicx < 1) // Make sure mosaic is valid
82         mosaicx = 1;
83     if(mosaicy < 1)
84         mosaicy = 1;
85
86     if(BMPScale < 0.05) {
87         BMPScale = 0.05f;
88         //printf("\n The scale was too small, was reset to 0.05\n");
89     }
90     if(BMPScale > 1.00) {
91         BMPScale = 1.0;
92         //printf("\n The scale was too large, was reset to 1.0");
93     }
94
95     //m_dpiX = GetDeviceCaps(hdc, LOGPIXELSX);
96     m_dpiY = GetDeviceCaps(hdc, LOGPIXELSY); // 96.0f;
97
98     ct = cos(glob_rot * GetPi() / 180.0f);
99     st = sin(glob_rot * GetPi() / 180.0f);
100    for (i = 0; i <= GetNBase(); i++) {
101        xr = GetBases()[i].x * ct + GetBases()[i].y * st;
102        yr = GetBases()[i].y * ct - GetBases()[i].x * st;
103        GetBases()[i].x = xr;
104        GetBases()[i].y = yr;
105    }
106    for (i = 0; i < GetLoopCount(); i++) {
107        lp = GetLoops() + i;
108        xr = lp->x * ct + lp->y * st;
109        yr = lp->y * ct - lp->x * st;
110        lp->x = xr;
111        lp->y = yr;
112    }
113    xmax = ymax = -GetBigNum();
114    xmin = ymin = GetBigNum();
115    for (i = 0; i <= GetNBase(); i++) {
116        if (GetBases()[i].x > GetANum() - 100.0) {
117            //printf("\nError -- base %d position is undefined.\n",i);
118        }
119        else {
120            xmax = maxf2(xmax, GetBases()[i].x);
121            ymax = maxf2(ymax, GetBases()[i].y);
122            xmin = minf2(xmin, GetBases()[i].x);
123            ymin = minf2(ymin, GetBases()[i].y);
124        }
125    }
126    scalex = (xmax - xmin) / xsize / mosaicx;
127    scaley = (ymax - ymin) / ysize / mosaicy;
128    scale = maxf2(scalex, scaley) * 1.2f;  // The 1.02 makes padding around the image
129
130    m_view_x = m_init_view_x / scale; // Set the view matrix, depends on naview input
131    m_view_y = m_init_view_y / scale;
```

```
132
133     // Scale the view matrix
134     if(BMPScale != 0.0) {
135         m_view_x = m_view_x / BMPScale;
136         m_view_y = m_view_y / BMPScale;
137     }
138     //else
139         //printf("\n Attempt to divide by zero when setting scale");
140
141     // Check to find offsets for centering
142     xs = xsize * scale;
143     ys = ysize * scale;
144     m_xoff = (xs * mosaicx - xmax + xmin) / 2.0f;
145     m_yoff = (ys * mosaicy - ymax + ymin) / 2.0f;
146
147     // Set the Font to scaled size
148     m_font_height = CalcFontHeight(csz, scale);
149
150     for (imx = 1; imx <= mosaicx; imx++) {
151         for (imy = 1; imy <= mosaicy; imy++) {
152             if (GetTitle()[0] != '\0') {
153                 if (imx == (mosaicx + 1) / 2 && imy == 1) {
154                     DrawTitle(hdc, GetTitle(), xsize, ysize);
155                 }
156             }
157             m_user_tran_x = xmin + (imx - 1) * xs - m_xoff; // Set user translation ↙
    vector
158             m_user_tran_y = ymin + (imy - 1) * ys - m_yoff;
159             xmx = xmin + imx * xs - m_xoff;
160             ymx = ymin + imy * ys - m_yoff;
161
162             m_current_color = RGB(0, 0, 0); // Set Color to Black
163
164             if (GetBases()[0].x != GetANum()) {
165                 DrawCircle(hdc, 0.0f, GetBases()[0].x, GetBases()[0].y); // Draw the ↙
    starting black circle
166             }
167
168             // SEQUENCE LINES
169             if (draw_bases)
170                 scalecsz = scale * csz * 1.8f;
171             else
172                 scalecsz = 0.0;
173             for (i = 0; i <= GetNBase() - 1; i++) {
174                 if (GetBases()[i].x != GetANum() && GetBases()[i + 1].x != GetANum()) ↙
    {
175                     if (okx(GetBases()[i].x) && oky(GetBases()[i].y) || okx(GetBases()↙
    [i + 1].x) && oky(GetBases()[i + 1].y)) {
176                         pRNA->GetPlanarPos()[i].PlanarX = user_to_screen_x(GetBases() ↙
    [i + 1].x); // record postions for line from matrix to planar graph
177                         pRNA->GetPlanarPos()[i].PlanarY = user_to_screen_y(GetBases() ↙
    [i + 1].y); // record postions for line from matrix to planar graph
178                         xn = GetBases()[i+1].x - GetBases()[i].x;
179                         yn = GetBases()[i+1].y - GetBases()[i].y;
180                         r = sqrt(xn * xn + yn * yn);
181                         if (r > scalecsz) {
182                             xn /= r;
183                             yn /= r;
184                             x1 = GetBases()[i].x + xn * scalecsz / 2.0f;
185                             y1 = GetBases()[i].y + yn * scalecsz / 2.0f;
186                             x2 = GetBases()[i+1].x - xn * scalecsz / 2.0f;
187                             y2 = GetBases()[i+1].y - yn * scalecsz / 2.0f;
188
189                             DrawLine(hdc, x1, y1, x2, y2); // Draw line
190                         }
191                     }
192                 }
```

```
193             }
194
195             // BASE PAIRING LINES WITH BASE PAIRS
196             #define draw_1bp \
197             if (dot_pairs) {\
198                 x1 = (GetBases()[i].x + GetBases()[mate].x) / 2.0f;\
199                 y1 = (GetBases()[i].y + GetBases()[mate].y) / 2.0f;\
200                 if (okx(x1) && oky(y1)) { \
201                     DrawCircle(hdc, 0.0, x1, y1); \
202                 } \
203             }\
204             else {\
205                 xn = GetBases()[mate].x - GetBases()[i].x;\
206                 yn = GetBases()[mate].y - GetBases()[i].y;\
207                 r = sqrt(xn * xn + yn * yn);\
208                 if (r > scalecsz) {\
209                     xn /= r;\
210                     yn /= r;\
211                     x1 = GetBases()[i].x + xn * scalecsz / 2.0f;\
212                     y1 = GetBases()[i].y + yn * scalecsz / 2.0f;\
213                     x2 = GetBases()[mate].x - xn * scalecsz / 2.0f;\
214                     y2 = GetBases()[mate].y - yn * scalecsz / 2.0f;\
215                     if (okx(x1) && oky(y1) || okx(x2) && oky(y2)) \
216                         DrawLine(hdc, x1, y1, x2, y2); \
217                 }\
218             }
219
220             m_current_color = RGB(255, 0, 0); // Set Color to Red
221             for (i = 0; i <= GetNBase(); i++) {
222                 if ((mate = GetBases()[i].mate) && i < mate ) {
223                     if (GetBases()[i].x != GetANum() && GetBases()[mate].x != GetANum ↙
    ()) {
224                         if (GetBases()[i].name == 'G' && GetBases()[mate].name == 'C' ↙
    || GetBases()[i].name == 'C' && GetBases()[mate].name == 'G') {
225                             draw_1bp;
226                         }
227                     }
228                 }
229             }
230
231             m_current_color = RGB(0, 0, 255); // Set Color to Blue
232             for (i = 0; i <= GetNBase(); i++) {
233                 if ((mate = GetBases()[i].mate) && i < mate ) {
234                     if (GetBases()[i].x != GetANum() && GetBases()[mate].x != GetANum ↙
    ()) {
235                         if (!(GetBases()[i].name == 'G' && GetBases()[mate].name == 'C↙
    ' || GetBases()[i].name == 'C' && GetBases()[mate].name == 'G')) {
236                             draw_1bp;
237                         }
238                     }
239                 }
240             }
241
242             m_current_color = RGB(0, 0, 0); // Set Color to Black
243             if (label_rate > 0) {
244                 // LABELS
245                 istart = label_rate * (1 + GetBases()[1].hist_num / label_rate) -     ↙
    GetBases()[1].hist_num + 1;
246                 for (i = istart; i <= GetNBase(); i += label_rate) {
247                     if (GetBases()[i].x != GetANum()) {
248                         if (okx(GetBases()[i].x) && oky(GetBases()[i].y)) {
249                             float dx, dy, angle;
250                             int ia;
251                             if (i == GetNBase()) {
252                                 dx = GetBases()[i].x - GetBases()[i-1].x;
253                                 dy = GetBases()[i].y - GetBases()[i-1].y;
254                             }
```

```
255                             else {
256                                 dx = GetBases()[i+1].x - GetBases()[i].x;
257                                 dy = GetBases()[i+1].y - GetBases()[i].y;
258                             }
259                             angle = atan2(dy, dx) / GetDTor() - 90.0f;
260                             ia = angle;
261                             DrawLabel(hdc, ia, GetBases()[i].x, GetBases()[i].y,
    GetBases()[i].hist_num);
262                         }
263                     }
264                 }
265             }
266 
267             if (mark_loops) {
268                 // LOOP LABELS
269                 for (i = 0; i < GetLoopCount(); i++) {
270                     lp = GetLoops() + i;
271                     if (okx(lp->x) && oky(lp->y))
272                         DrawLoopNumber(hdc, lp->x, lp->y, lp->number);
273                 }
274             }
275 
276             if (draw_bases) {
277                 // BASES
278                 for (i = 1; i <= GetNBase(); i++) {
279                     if (GetBases()[i].x != GetANum()) {
280                         if (okx(GetBases()[i].x) && oky(GetBases()[i].y)) {
281                             DrawBase(hdc, GetBases()[i].x, GetBases()[i].y, GetBases()
    [i].name);
282                         }
283                     }
284                 }
285             }
286         }
287     }
288 }
289 
290 // FUNCTION:  CalcFontHeight
291 // Used to scale the font size
292 //
293 // @param   char_size   float   the character size
294 // @param   scale       float   the scale factor
295 // @return  float
296 float RNAGraphBMP::CalcFontHeight(float char_size, float scale) {
297     float nHeight; // in points
298 
299     //nHeight = -MulDiv(PointSize, GetDeviceCaps(hDC, LOGPIXELSY), 72);
300     nHeight = ((char_size * m_dpiY) / 72) / scale;
301     if(nHeight > 29.0) // Put an upper bound on the font size
302         nHeight = 29.0;
303 
304     return nHeight;
305 }
306 
307 /*
308 * Remove quotes and remove line feeds
309 */
310 // FUNCTION:  remove_quotes
311 // Removes quotes and removes line feeds
312 //
313 // @param   string    char*   the string to remove the characters from
314 // @return  void
315 void RNAGraphBMP::remove_quotes(char *string) {
316     int i, len;
317     len = (int)strlen(string);
318     for(i = 0; i < len; i++) {
319         if(string[i] == '"') {
```

```
320             string[i] = ' ';
321             if(i == (len - 1))
322                 string[i] = '\0';
323         }
324         if(string[i] == 10)
325             string[i] = '\0';
326     }
327 }
328
329 // FUNCTION:  DrawTitle
330 // Draw title on bottom center of device context
331 //
332 // @param   hdc        HDC     handle to bitmap device context
333 // @param   title      char*   the title character array
334 // @param   BMPSizeX   int     the x dimension of the bitmap
335 // @param   BMPSizeY   int     the y dimension of the bitmap
336 // @return  void
337 void RNAGraphBMP::DrawTitle(HDC hdc, char *title, int BMPSizeX, int BMPSizeY) {
338     int nHeight = 15.0f;
339     remove_quotes(title); // remove unwanted characters
340
341     HFONT hFont = CreateFont(nHeight, 0, 0, 0, FW_SEMIBOLD, FALSE, FALSE, FALSE,
        ANSI_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
        FF_DONTCARE, TEXT("Arial"));
342     HFONT hOldFont = (HFONT)SelectObject(hdc, hFont);
343
344     int len = strlen(title);
345     SIZE sizeRect;
346     GetTextExtentPointA(hdc, title, len, &sizeRect);
347     TextOutA(hdc, (BMPSizeX / 2) - (sizeRect.cx / 2), BMPSizeY - sizeRect.cy - 10,
        title, len);
348
349     SelectObject(hdc, hOldFont);
350     DeleteObject(hFont);
351 }
352
353 // FUNCTION:  DrawLine
354 // Draw a line in user coordinates from x1,y1 to x2,y
355 //
356 // @param   hdc    HDC     handle to bitmap device context
357 // @param   x1     float   start x coordinate
358 // @param   y1     float   start y coordinate
359 // @param   x2     float   end x coordinate
360 // @param   y2     float   end y coordinate
361 // @return  void
362 void RNAGraphBMP::DrawLine(HDC hdc, float x1, float y1, float x2, float y2) {
363     float bmp_x1, bmp_x2, bmp_y1, bmp_y2;
364     float line_width;
365
366     bmp_x1 = user_to_screen_x(x1);
367     bmp_x2 = user_to_screen_x(x2);
368     bmp_y1 = user_to_screen_y(y1);
369     bmp_y2 = user_to_screen_y(y2);
370
371     if(m_current_color == RGB(0, 0, 0))
372         line_width = 1.0f; // 0.69f * m_view_x + 0.1f; // set width for black lines
373     else
374         line_width = 2.0f; // 1.2f * m_view_x + 0.15f; // set width for non black
        lines
375
376     if(m_line_width != line_width) {
377         m_line_width = line_width;
378    }
379
380     HPEN hPen = CreatePen(PS_SOLID, m_line_width, m_current_color);
381     HPEN hOldPen = (HPEN)SelectObject(hdc, hPen);
382     MoveToEx(hdc, (int)bmp_x1, (int)bmp_y1, NULL);
```

```
383     LineTo(hdc, (int)bmp_x2, (int)bmp_y2);
384     SelectObject(hdc, hOldPen);
385     DeleteObject(hPen);
386 }
387
388 // FUNCTION:  DrawCircle
389 // Draws a circle at current user coordinates, radius of zero is converted to look
        nice
390 //
391 // @param   hcd     HDC    handle to bitmap device context
392 // @param   radius  float  radius of circle
393 // @param   x       float  x coordinate of circle center
394 // @param   y       float  y coordinate of circle center
395 // @return  void
396 void RNAGraphBMP::DrawCircle(HDC hdc, float radius, float x, float y) {
397     float bmp_x, bmp_y, radius_in_p;
398
399     bmp_x = user_to_screen_x(x);
400     bmp_y = user_to_screen_y(y);
401
402     // basepair circles result from zero radius
403     if(radius != 0.0) {
404         radius_in_p = (m_dpiY * radius * m_view_x);
405         MoveToEx(hdc, (int)bmp_x, (int)bmp_y, NULL);
406     }
407     else // set radius of circle
408         radius_in_p = (m_dpiY * 0.002f * m_view_x);
409
410     // set a maximum on radius size
411     if(radius_in_p > 20) {
412         radius_in_p = 20;
413     }
414
415     // fprintf(psfp,"%.2f %.2f %.2f 0 360 arc closepath fill \n", ps_x,ps_y,
        radius_in_p);
416     //HPEN hPen = CreatePen(PS_SOLID, m_line_width, m_current_color);
417     //HPEN hOldPen = (HPEN)SelectObject(hdc, hPen);
418     HBRUSH hBrush = CreateSolidBrush(m_current_color);
419     HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);
420     Ellipse(hdc, (int)(bmp_x - radius_in_p), (int)(bmp_y - radius_in_p), (int)(bmp_x +
         radius_in_p), (int)(bmp_y + radius_in_p));
421     SelectObject(hdc, hOldBrush);
422     DeleteObject(hBrush);
423     //SelectObject(hdc, hOldPen);
424     //DeleteObject(hPen);
425 }
426
427 // FUNCTION:  DrawLabel
428 // Draw string with left character at x,y in user coordinates given user coordinates
429 //
430 // @param   hdc    HDC    handle to bitmap device context
431 // @param   angle  float  the current angle
432 // @param   x      float  the x coordinate
433 // @param   y      float  the y coordinate
434 // @param   number int    the number to darw
435 // @return  void
436 void RNAGraphBMP::DrawLabel(HDC hdc, float angle, float x, float y, int number) {
437     float x1, y1; // location to place label
438     float bmp_sx, bmp_x1, bmp_x2;
439     float bmp_sy, bmp_y1, bmp_y2;
440     float line_width;
441
442     int len;
443     char strNumber[15];
444     itoa(number, strNumber, 10);
445     len = strlen(strNumber);
446
```

```
447     // place label m_line_length away based on angle
448     x1 = x + m_line_length * (float)cos((double)(angle * -3.14159 / 180.0)); //
        convert to radians
449     y1 = y + m_line_length * (float)sin((double)(angle * -3.14159 / 180.0));
450     bmp_sx = user_to_screen_x(x1);
451     bmp_sy = user_to_screen_y(y1);
452     x1 = x + 0.65f * m_line_length * (float)cos((double)(angle * -3.14159 / 180.0)); /
        / convert to radians
453     y1 = y + 0.65f * m_line_length * (float)sin((double)(angle * -3.14159 / 180.0));
454     bmp_x1 = user_to_screen_x(x1);
455     bmp_y1 = user_to_screen_y(y1);
456     x1 = x + 0.25f * m_line_length * (float)cos((double)(angle * -3.14159 / 180.0)); /
        / convert to radians
457     y1 = y + 0.25f * m_line_length * (float)sin((double)(angle * -3.14159 / 180.));
458     bmp_x2 = user_to_screen_x(x1);
459     bmp_y2 = user_to_screen_y(y1);
460     line_width = 1.0f; // 0.4f * m_view_x * + 0.10f; // sets line  width for labels
461     if(m_line_width != line_width) {
462         m_line_width = line_width;
463     }
464
465     HFONT hFont = CreateFont(m_font_height, 0, 0, 0, FW_THIN, FALSE, FALSE, FALSE,
        ANSI_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
        FF_DONTCARE, TEXT("Arial"));
466     HFONT hOldFont = (HFONT)SelectObject(hdc, hFont);
467     SIZE objSize;
468     GetTextExtentPointA(hdc, strNumber, len, &objSize);
469     TextOutA(hdc, bmp_sx - (objSize.cx / 2), bmp_sy - (objSize.cy / 2), strNumber,
        len);
470     SelectObject(hdc, hOldFont);
471     DeleteObject(hFont);
472
473     HPEN hPen = CreatePen(PS_SOLID, m_line_width, m_current_color);
474     HPEN hOldPen = (HPEN)SelectObject(hdc, hPen);
475     MoveToEx(hdc, (int)bmp_x1, (int)bmp_y1, NULL);
476     LineTo(hdc, (int)bmp_x2, (int)bmp_y2);
477     SelectObject(hdc, hOldPen);
478     DeleteObject(hPen);
479 }
480
481 // FUNCTION:  DrawCenteredText
482 // Draw text centered at user coordinates x,y.  This code is used for most
483 // text.  Converts to screen coordinates
484 //
485 // @param   hdc    HDC    the bitmap device context
486 // @param   x      float  the x coordinate
487 // @param   y      float  the y coordinate
488 // @param   string char*  the string to draw
489 // @return  void
490 void RNAGraphBMP::DrawCenteredText(HDC hdc, float x, float y, char *string) {
491     float screen_x, screen_y;
492
493     screen_x = user_to_screen_x(x);
494     screen_y = user_to_screen_y(y);
495
496     HFONT hFont = CreateFont(m_font_height, 0, 0, 0, FW_THIN, FALSE, FALSE, FALSE,
        ANSI_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
        FF_DONTCARE, TEXT("Arial"));
497     HFONT hOldFont = (HFONT)SelectObject(hdc, hFont);
498     int len = strlen(string);
499     SIZE objSize;
500     GetTextExtentPointA(hdc, string, len, &objSize);
501     TextOutA(hdc, screen_x - (objSize.cx / 2), screen_y - (objSize.cy / 2), string,
        len);
502     SelectObject(hdc, hOldFont);
503     DeleteObject(hFont);
504 }
```

```
505
506 // FUNCTION:  DrawBase
507 // Used to Draw the base symbol
508 //
509 // @param   hdc    HDC    handle to bitmap device context
510 // @param   x      float  x coordinate
511 // @param   y      float  y coordinate
512 // @param   base   char   base symbol
513 // @return  void
514 void RNAGraphBMP::DrawBase(HDC hdc, float x, float y, char base) {
515     char strBase[2];
516     strBase[0] = base;
517     strBase[1] = '\0';
518     DrawCenteredText(hdc, x, y, strBase);
519 }
520
521 // FUNCTION:  DrawLoopNumber
522 // Used to draw the loop number
523 //
524 // @param   hdc    HDC    handle to bitmap device context
525 // @param   x      float  the x coordinate
526 // @param   y      float  the y coordinate
527 // @param   number int    the number to darw
528 // @return  void
529 void RNAGraphBMP::DrawLoopNumber(HDC hdc, float x, float y, int number) {
530     char strNumber[15];
531     itoa(number, strNumber, 10);
532     DrawCenteredText(hdc, x, y, strNumber);
533 }
534
535 // FUNCTION:  user_to_screen_x
536 // Used to convert the x coordinate from user to screen coordinates
537 //
538 // @param   x   float   the number to convert
539 // @return  float
540 float RNAGraphBMP::user_to_screen_x(float x) {
541     float x_screen;
542     x_screen = m_view_x * (x - m_user_tran_x) + m_origin_x;
543     return x_screen;
544 }
545
546 // FUNCTION:  user_to_screen_y
547 // Used to convert the y coordinate from user to screen coordinates
548 //
549 // @param   y    float   the number to convert
550 // @return  float
551 float RNAGraphBMP::user_to_screen_y(float y) {
552     float y_screen;
553     y_screen = m_view_y * (y - m_user_tran_y) + m_origin_y;
554     return y_screen;
555 }
```

```
/*****************************************************
* Copyright (c): 2006, All Rights Reserved
* Project:       SJSU Masters Project
* File:          RNAGraph.cpp
* Purpose:       To calculate a graph of RNA secondary structure
*
* Start Date:    8/1/2006
* Programmer:    Brandon Hunter
*
*****************************************************
*
* This program is based on a program called NAVIEW written
* by Robert E. Bruccoleri and Michael Zuker
*
*   NAVIEW -- A program to make a modified radial drawing of an RNA
*   secondary structure.
*
*   Copyright (c) 1988 Robert E. Bruccoleri
*   Copying of this software, in whole or in part, is permitted
*   provided that the copies are not made for commercial purposes,
*   appropriate credit for the use of the software is given, this
*   copyright notice appears, and notice is given that the copying
*   is by permission of Robert E. Bruccoleri. Any other copying
*   requires specific permission.
*
*   See R. Bruccoleri and G. Heinrich, Computer Applications in the
*   Biosciences, 4, 167-173 (1988) for a full description.
*/

#pragma once

class RNAGraph
{
public:
    #define maxiter 500
    #define type_alloc(type) (type *) malloc(sizeof(type))
    #define struct_alloc(structure_name) type_alloc(struct structure_name)

    // Forward Declarations
    typedef struct region_tag;
    typedef struct connection_tag;
    typedef struct loop_tag;
    typedef struct base_tag;
    typedef struct radloop_tag;

    typedef struct region_tag {
        int start1, end1, start2, end2;
    }Region;

    typedef struct connection_tag {
        struct loop_tag *loop;
        struct region_tag *region;
        int start, end; // Start and end form the 1st base pair of the region.
        float xrad, yrad, angle;
        bool extruded;  // True if segment between this connection and
                        // the next must be extruded out of the circle
        bool broken;    // True if the extruded segment must be drawn long.
    }Connection;

    typedef struct loop_tag {
        int nconnection;
        struct connection_tag **connections;
        int number;
        int depth;
        bool mark;
        float x, y, radius;
    }Loop;
```

```
68
69      typedef struct base_tag {
70          char name;
71          int mate, hist_num;
72          float x, y;
73          bool extracted;
74          struct region_tag *region;
75      }Base;
76
77      typedef struct radloop_tag {
78          float radius;
79          int loopnumber;
80          struct radloop_tag *next, *prev;
81      }Radloop;
82
83  private:
84      float m_pi;
85      float m_dtor;
86      float m_bignum;
87      float m_anum;
88
89      float m_minbaseseparation;
90      struct Base *bases;
91      int nbase, nregion, loop_count;
92      struct Loop *anchor, *root, *loops;
93      struct Region *regions;
94      struct Radloop *rlphead, *rlptail;
95
96      #define LINEMX 500
97      char line[LINEMX], title[LINEMX];
98
99      //struct loop_tag *construct_loop();
100     void CalcGraph();
101     void find_regions();
102     void dump_loops();
103     void find_central_loop();
104     bool connected_connection(struct Connection *cp, struct Connection *cpnext);
105     int find_ic_middle(int icstart, int icend, struct Connection *anchor_connection,
        struct Connection *acp, struct Loop *lp);
106     void determine_depths();
107     int depth(struct Loop *lp);
108     void traverse_loop(struct Loop *lp, struct Connection *anchor_connection);
109     void determine_radius(struct Loop *lp, float lencut);
110     void generate_region(struct Connection *cp);
111     void construct_extruded_segment(struct Connection *cp, struct Connection *cpnext);
112     void construct_circle_segment(int start, int end);
113     void find_center_for_arc(int n, float b, float *hp, float *thetap);
114     void trim(char *st, int *stlenp);
115     void chcnbl(char *st, int stlen);
116     struct loop_tag *construct_loop(int ibase);
117     void FreeMemory(void);
118     void CalcSeqeuceWithoutLoops(void);
119
120 public:
121     RNAGraph(float MinBaseSeparation);
122     ~RNAGraph(void);
123
124     void LoadFromCTFile(char *inputfile);
125     void LoadFromRNASequence(char* Title, char* Sequence, int Length, int* Pairing);
126     void Write_PLT2_Output(char *dev, char *outputfile, float xsize, float ysize, bool
         mark_loops, bool draw_bases, float csz, int label_rate, bool dot_pairs, int
        mosaicx, int mosaicy, float glob_rot);
127
128     float minf2(float x1, float x2);
129     float maxf2(float x1, float x2);
130
131     void SetMinBaseSeparation(float MinBaseSeparation);
```

```
132     float GetMinBaseSeparation(void);
133     void ChangeLoopRadii();
134     float GetPi(void);
135     struct base_tag *GetBases(void);
136     int GetNBase(void);
137     int GetLoopCount(void);
138     struct loop_tag *GetLoops();
139     float GetBigNum(void);
140     float GetANum(void);
141     char *GetTitle(void);
142     float GetDTor(void);
143 };
```

```
/*****************************************************
* Copyright (c): 2006, All Rights Reserved
* Project:       SJSU Masters Project
* File:          RNAGraph.cpp
* Purpose:       To calculate a graph of RNA secondary structure
*
* Start Date:    8/1/2006
* Programmer:    Brandon Hunter
*
*****************************************************
*
* This program is based on a program called NAVIEW written
* by Robert E. Bruccoleri
*
*   NAVIEW -- A program to make a modified radial drawing of an RNA
*   secondary structure.
*
*   Copyright (c) 1988 Robert E. Bruccoleri
*   Copying of this software, in whole or in part, is permitted
*   provided that the copies are not made for commercial purposes,
*   appropriate credit for the use of the software is given, this
*   copyright notice appears, and notice is given that the copying
*   is by permission of Robert E. Bruccoleri. Any other copying
*   requires specific permission.
*
*   See R. Bruccoleri and G. Heinrich, Computer Applications in the
*   Biosciences, 4, 167-173 (1988) for a full description.
*********************************************************************
*
* Modified August 2006 by Brad Hunter.
* Turned the code into a class
* Modified code so that small sequences with no loops are handled
*/

#include "StdAfx.h"
#include "RNAGraph.h"

#include <math.h>
#include <stdio.h>

// FUNCTION:  RNAGraph
// Default Constructor
//
// @param   MinBaseSeparation   float  the minimum base separation
// @return  void
RNAGraph::RNAGraph(float MinBaseSeparation) {
    SetMinBaseSeparation(MinBaseSeparation);

    m_pi = 3.141592653589793f;
    m_dtor = (3.141592653589793f / 180.0f);
    m_bignum = 1.7e38f;
    m_anum = 9999.0f;

    bases = NULL;
    regions = NULL;
    loops = NULL;
    rlphead = rlptail = NULL; // Initialize the loop radius linked list
}

// FUNCTION:  ~RNAGraph
// Default Destructor
//
// @param   undefined   void
// @return  void
RNAGraph::~RNAGraph(void) {
    FreeMemory(); // Free the dynamically allocated memory
}
```

```
 68
 69 // FUNCTION:  FreeMemory
 70 // Used to free memory that was dynamically allocated
 71 //
 72 // @param   undefined   void
 73 // @return  void
 74 void RNAGraph::FreeMemory(void) {
 75     if(bases != NULL) {
 76         free(bases);
 77         bases = NULL;
 78     }
 79     if(regions != NULL) {
 80         free(regions);
 81         regions = NULL;
 82     }
 83
 84     if(loops != NULL) {
 85         for(int x = 0; x < loop_count; x++) {
 86             for(int y = 0; y < loops[x].nconnection; y++) {
 87                 free(*(loops[x].connections + y));
 88             }
 89             free(loops[x].connections);
 90         }
 91         free(loops);
 92         loops = NULL;
 93     }
 94
 95     // Free the radloop list
 96     struct Radloop *rlp = rlphead;
 97     while(rlp) {
 98         rlp = rlp->next;
 99         free(rlphead);
100         rlphead = rlp;
101     }
102 }
103
104 // FUNCTION:  LoadFromRNASequence
105 // Used to load the sequence into the class
106 //
107 // @param   Title     char*   the Title of the sequence
108 // @param   Sequence  char*   the sequence of nucleotide bases
109 // @param   Length    int     the length of the sequence
110 // @param   Pairing   int*    an array of integers that defines the pairing
111 // @return  void
112 void RNAGraph::LoadFromRNASequence(char* Title, char* Sequence, int Length, int*
    Pairing) {
113     FreeMemory(); // Free Memory From Previous Calculation
114
115     nbase = Length;
116
117     strcpy(title, Title);
118
119     bases = (struct Base *) malloc(sizeof(struct Base) * (Length + 1));
120
121     // Set up an origin
122     bases[0].name = 'o';
123     bases[0].mate = 0;
124     bases[0].hist_num = 0;
125     bases[0].extracted = false;
126     bases[0].x = m_anum;
127     bases[0].y = m_anum;
128     for(int i = 1; i <= Length; i++) {
129         bases[i].name = Sequence[i - 1];
130         if(Pairing[i - 1] == i - 1)
131             bases[i].mate = 0;
132         else
133             bases[i].mate = Pairing[i - 1] + 1;
```

```
134         bases[i].hist_num = i + 1;
135         bases[i].extracted = false;
136         bases[i].x = m_anum;
137         bases[i].y = m_anum;
138     }
139 
140     CalcGraph();
141 }
142 
143 // FUNCTION:  LoadFromCTFile
144 // Used to load the sequence from a CT-File
145 //
146 // @param   inputfile  char*   the file path to where the ct file exists
147 // @return  void
148 void RNAGraph::LoadFromCTFile(char *inputfile) {
149     FILE *inf;
150     char *cp;
151     int i, linel;
152 
153     FreeMemory(); // Free Memory From Previous Calculation
154 
155     inf = fopen(inputfile, "r");
156     if (inf == NULL) {
157         throw "RNAGraph: Unable to open input file";
158     }
159     if ((cp = fgets(line, LINEMX, inf)) == NULL) {
160         throw "RNAGraph: Unable to read header record";
161     }
162 
163     line[76] = '\0';
164     linel = strlen(line);
165     chcnbl(line, linel);
166     trim(line, &linel);
167     line[linel] = '\0';
168     sscanf(line, "%5d", &nbase);
169     //printf("%d bases specified in file. Title is \n%s\n", nbase, &line[5]);
170     strcpy(title, &line[5]);
171 
172     bases = (struct Base *) malloc(sizeof(struct Base) * (nbase + 1));
173 
174     // Set up an origin
175     bases[0].name = 'o';
176     bases[0].mate = 0;
177     bases[0].hist_num = 0;
178     bases[0].extracted = false;
179     bases[0].x = m_anum;
180     bases[0].y = m_anum;
181     for(i = 1; i <= nbase; i++) {
182         if((cp = fgets(line, LINEMX, inf)) == NULL)
183             break;
184 
185         sscanf(line,"%*d %c %*d %*d %d %d",&bases[i].name,&bases[i].mate, &bases[i]. ↙
    hist_num);
186         // The code in this program depends on mate being false (namely zero) where  ↙
    there is no mate.
187         bases[i].extracted = false;
188         bases[i].x = m_anum;
189         bases[i].y = m_anum;
190     }
191     if(--i != nbase) {
192         throw "RNAGraph: Number of bases read from file, doesn't match CT File header↙
     record";
193         nbase = i;
194     }
195 
196     CalcGraph();
197 }
```

```
198
199 // FUNCTION:  CalcGraph
200 // Used to calculate the secondary structure
201 //
202 // @param   undefined   void
203 // @return  void
204 void RNAGraph::CalcGraph()
205 {
206     // Constructing tree of loops
207     nregion = 0;
208     loop_count = 0;
209     regions = (struct Region *) malloc(sizeof(struct Region) * (nbase + 1));
210     find_regions();
211     if(nregion == 0) {
212         CalcSeqeuceWithoutLoops();
213     }
214     else {
215         loops = (struct Loop *) malloc(sizeof(struct Loop) * (nbase + 1));
216         anchor = construct_loop(0);
217         find_central_loop();
218         //if (debug) dump_loops();
219         traverse_loop(root, NULL); // Constructing the drawing
220     }
221 }
222
223
224 // FUNCTION:  find_regions
225 // Identifies the regions in the structure
226 //
227 // @param   undefined   void
228 // @return  void
229 void RNAGraph::find_regions() {
230     int i, mate, nb1;
231     bool *mark;
232
233     nb1 = nbase + 1;
234     mark = (bool *) malloc(sizeof(int) * nb1);
235     for (i = 0; i < nb1; i++)
236         mark[i] = false;
237     nregion = 0;
238     for(i = 0; i <= nbase; i++) {
239         if((mate = bases[i].mate) && !mark[i]) {
240             regions[nregion].start1 = i;
241             regions[nregion].end2 = mate;
242             mark[i] = true;
243             mark[mate] = true;
244             bases[i].region = bases[mate].region = &regions[nregion];
245             for(i++, mate--; i < mate && bases[i].mate == mate; i++, mate--) {
246                 mark[i] = mark[mate] = true;
247                 bases[i].region = bases[mate].region = &regions[nregion];
248             }
249             regions[nregion].end1 = --i;
250             regions[nregion].start2 = mate+1;
251             //if(debug) {
252             //  if(nregion == 0) printf("\nRegions are:\n");
253             //      printf("Region %d is %d-%d and %d-%d with gap of %d.\n",
254             //          nregion+1,regions[nregion].start1,regions[nregion].end1,
255             //          regions[nregion].start2,regions[nregion].end2,
256             //          regions[nregion].start2-regions[nregion].end1+1);
257             //}
258             nregion++;
259         }
260     }
261     free(mark);
262 }
263
264 // FUNCTION:  construct_loop
```

```
265 // Starting at residue ibase, recursively construct the loop containing
266 // said base and all deeper bases
267 //
268 // @param   ibase   int   the base location
269 // @return  struct loop_tag*
270 struct RNAGraph::loop_tag *RNAGraph::construct_loop(int ibase) {
271     int i, mate;
272     struct Loop *retloop, *lp;
273     struct Connection *cp;
274     struct Region *rp;
275     struct Radloop *rlp;
276 
277     retloop = &loops[loop_count++];
278     retloop->nconnection = 0;
279     retloop->connections = (struct Connection **) malloc(sizeof(struct Connection *))
    ;
280     retloop->depth = 0;
281     retloop->number = loop_count;
282     retloop->radius = 0.0;
283     for (rlp = rlphead;  rlp;  rlp = rlp->next)
284         if (rlp->loopnumber == loop_count)
285             retloop->radius = rlp->radius;
286     i = ibase;
287     do {
288         if ((mate = bases[i].mate) != 0) {
289             rp = bases[i].region;
290             if (!bases[rp->start1].extracted) {
291                 if (i == rp->start1) {
292                     bases[rp->start1].extracted = true;
293                     bases[rp->end1].extracted = true;
294                     bases[rp->start2].extracted = true;
295                     bases[rp->end2].extracted = true;
296                     lp = construct_loop(rp->end1 < nbase ? rp->end1+1 : 0);
297                 }
298                 else if (i == rp->start2){
299                     bases[rp->start2].extracted = true;
300                     bases[rp->end2].extracted = true;
301                     bases[rp->start1].extracted = true;
302                     bases[rp->end1].extracted = true;
303                     lp = construct_loop(rp->end2 < nbase ? rp->end2+1 : 0);
304                 }
305                 else {
306                     throw "RNAGraph: Error detected in construct_loop. i = %d not
    found in region table";
307                 }
308                 retloop->connections = (struct Connection **)realloc(retloop->
    connections, (++retloop->nconnection + 1) * sizeof(struct Connection *));
309                 retloop->connections[retloop->nconnection - 1] = cp = struct_alloc
    (Connection);
310                 retloop->connections[retloop->nconnection] = NULL;
311                 cp->loop = lp;
312                 cp->region = rp;
313                 if (i == rp->start1) {
314                     cp->start = rp->start1;
315                     cp->end = rp->end2;
316                 }
317                 else {
318                     cp->start = rp->start2;
319                     cp->end = rp->end1;
320                 }
321                 cp->extruded = false;
322                 cp->broken = false;
323                 lp->connections = (struct Connection **) realloc(lp->connections, (++
    lp->nconnection + 1) * sizeof(struct Connection *));
324                 lp->connections[lp->nconnection-1] = cp = struct_alloc(Connection);
325                 lp->connections[lp->nconnection] = NULL;
326                 cp->loop = retloop;
```

```
327                 cp->region = rp;
328                 if (i == rp->start1) {
329                     cp->start = rp->start2;
330                     cp->end = rp->end1;
331                 }
332                 else {
333                     cp->start = rp->start1;
334                     cp->end = rp->end2;
335                 }
336                 cp->extruded = false;
337                 cp->broken = false;
338             }
339             i = mate;
340         }
341         if (++i > nbase) i = 0;
342     }
343     while (i != ibase);
344 
345     return retloop;
346 }
347 
348 // FUNCTION:  dump_loops
349 // Displays all theh loops
350 //
351 // @param   undefined   void
352 // @return  void
353 void RNAGraph::dump_loops() {
354     int il,ilp,irp;
355     struct Loop *lp;
356     struct Connection *cp,**cpp;
357 
358     //printf("\nRoot loop is #%d\n",(root-loops)+1);
359     for (il=0; il < loop_count; il++) {
360         lp = &loops[il];
361         //printf("Loop %d has %d connections:\n",il+1,lp->nconnection);
362         for (cpp = lp->connections; cp = *cpp; cpp++) {
363             ilp = (cp->loop - loops) + 1;
364             irp = (cp->region - regions) + 1;
365             //printf("  Loop %d Region %d (%d-%d)\n", ilp,irp,cp->start,cp->end);
366         }
367     }
368 }
369 
370 // FUNCTION:  find_central_loop
371 // Find node of greatest branching that is deepest
372 //
373 // @param   undefined   void
374 // @return  void
375 void RNAGraph::find_central_loop() {
376     struct Loop *lp;
377     int maxconn, maxdepth, i;
378 
379     determine_depths();
380     maxconn = 0;
381     maxdepth = -1;
382 
383     for(i = 0; i < loop_count; i++) {
384         lp = &loops[i];
385         if(lp->nconnection > maxconn) {
386             maxdepth = lp->depth;
387             maxconn = lp->nconnection;
388             root = lp;
389         }
390         else if(lp->depth > maxdepth && lp->nconnection == maxconn) {
391             maxdepth = lp->depth;
392             root = lp;
393         }
```

```
394     }
395 }
396
397 // FUNCTION:  determine_depths
398 // Determine the depth of all loops
399 //
400 // @param   undefined   void
401 // @return  void
402 void RNAGraph::determine_depths() {
403     struct Loop *lp;
404     int i, j;
405
406     for(i = 0; i < loop_count; i++) {
407         lp = &loops[i];
408         for(j = 0; j < loop_count; j++)
409             loops[j].mark = false;
410         lp->depth = depth(lp);
411     }
412 }
413
414 // FUNCTION:  depth
415 // Determines the depth of loop, lp. Depth is defined as the minimum
416 // distance to a leaf loop where a leaf loop is one that has only one
417 // or no connections.
418 //
419 // @param   lp    Struct Loop*
420 // @return  int
421 int RNAGraph::depth(struct Loop *lp) {
422     struct Connection *cp, **cpp;
423     int count, ret, d;
424
425     if(lp->nconnection <= 1) return 0;
426     if(lp->mark) return -1;
427     lp->mark = true;
428     count = 0;
429     ret = 0;
430     for(cpp = lp->connections; cp = *cpp; cpp++) {
431         d = depth(cp->loop);
432         if(d >= 0) {
433             if(++count == 1) ret = d;
434             else if(ret > d) ret = d;
435         }
436     }
437     lp->mark = false;
438     return ret + 1;
439 }
440
441 // FUNCTION:  CalcSequenceWithoutLoops
442 // If the rna sequence doesn't have any loops then this function
443 // calculates the diagram of the sequence without loops
444 //
445 // @param   undefined   void
446 // @return  void
447 void RNAGraph::CalcSeqeuceWithoutLoops(void) {
448     float dt, angleinc, angleadjust, sumn, sumd, radius;
449
450     dt = 2.0f * m_pi;
451     angleinc = dt / (nbase + 1);
452     sumn = dt * (1.0f / nbase + 1.0f);
453     sumd = dt * dt / nbase;
454     radius = sumn / sumd;
455
456     angleadjust = 0.5f * m_pi;
457
458     for(int i = 0; i <= nbase; i++) {
459         bases[i].x = radius * cos(angleadjust + (angleinc * i));
460         bases[i].y = radius * sin(angleadjust + (angleinc * i));
```

```
461     }
462 }
463
464 // FUNCTION:  traverse_loop
465 // This is the workhorse of the display program. The algorithm is
466 // recursive based on processing individual loops. Each base pairing
467 // region is displayed using the direction given by the circle diagram,
468 // and the connections between the regions is drawn by equally spaced
469 // points. The radius of the loop is set to minimize the square error
470 // for lengths between sequential bases in the loops. The "correct"
471 // length for base links is 1. If the least squares fitting of the
472 // radius results in loops being less than 1/2 unit apart, then that
473 // segment is extruded.
474 //
475 // The variable, anchor_connection, gives the connection to the loop
476 // processed in an previous level of recursion.
477 //
478 // @param   lp                 struct Loop*          a pointer to the loop
479 // @param   anchor_connecton   struct Connection*    a pointer to the connection
480 // @return  void
481 void RNAGraph::traverse_loop(struct Loop *lp, struct Connection *anchor_connection) {
482     float xs, ys, xe, ye, xn, yn, angleinc, r;
483     float radius, xc, yc, xo, yo, astart, aend, a;
484     struct Connection *cp, *cpnext, **cpp, *acp, *cpprev;
485     int i, j, n, ic;
486     float da, maxang;
487     int count, icstart, icend, icmiddle, icroot;
488     bool done, done_all_connections, rooted;
489     int sign;
490     float midx, midy, nrx, nry, mx, my, vx, vy, dotmv, nmidx, nmidy;
491     int icstart1, icup, icdown, icnext, direction;
492     float dan, dx, dy, rr;
493     float cpx, cpy, cpnextx, cpnexty, cnx, cny, rcn, rc, lnx, lny, rl, ac, acn, sx,
      sy, dcp;
494     int imaxloop;
495
496     angleinc = 2 * m_pi / (nbase + 1);
497     acp = NULL;
498     icroot = -1;
499     for(cpp = lp->connections, ic = 0; cp = *cpp; cpp++, ic++) {
500         // xs = cos(angleinc*cp->start);
501         // ys = sin(angleinc*cp->start);
502         // xe = cos(angleinc*cp->end);
503         // ye = sin(angleinc*cp->end);
504         xs = -sin(angleinc * cp->start);
505         ys = cos(angleinc * cp->start);
506         xe = -sin(angleinc * cp->end);
507         ye = cos(angleinc * cp->end);
508         xn = ye - ys;
509         yn = xs - xe;
510         r = sqrt(xn * xn + yn * yn);
511         cp->xrad = xn / r;
512         cp->yrad = yn / r;
513         cp->angle = atan2(yn, xn);
514         if(cp->angle < 0.0) cp->angle += 2 * m_pi;
515         if(anchor_connection != NULL && anchor_connection->region == cp->region) {
516             acp = cp;
517             icroot = ic;
518         }
519     }
520
521 set_radius:
522     determine_radius(lp, m_minbaseseparation);
523     radius = lp->radius;
524     if(anchor_connection == NULL)
525         xc = yc = 0.0;
526     else {
```

```
527         xo = (bases[acp->start].x + bases[acp->end].x) / 2.0f;
528         yo = (bases[acp->start].y + bases[acp->end].y) / 2.0f;
529         xc = xo - radius * acp->xrad;
530         yc = yo - radius * acp->yrad;
531     }
532
533     /*
534     *   The construction of the connectors will proceed in blocks of
535     *   connected connectors, where a connected connector pairs means
536     *   two connectors that are forced out of the drawn circle because they
537     *   are too close together in angle.
538     */
539
540     // First, find the start of a block of connected connectors
541
542     if(icroot == -1)
543         icstart = 0;
544     else icstart = icroot;
545         cp = lp->connections[icstart];
546     count = 0;
547     //if (debug) printf("Now processing loop %d\n",lp->number);
548     done = false;
549     do {
550         j = icstart - 1;
551         if(j < 0) j = lp->nconnection - 1;
552         cpprev = lp->connections[j];
553         if(!connected_connection(cpprev,cp)) {
554             done = true;
555         }
556         else {
557             icstart = j;
558             cp = cpprev;
559         }
560         if(++count > lp->nconnection) {
561             // Here everything is connected. Break on maximum angular separation
562             // between connections.
563             maxang = -1.0;
564             for(ic = 0;  ic < lp->nconnection;  ic++) {
565                 j = ic + 1;
566                 if(j >= lp->nconnection) j = 0;
567                 cp = lp->connections[ic];
568                 cpnext = lp->connections[j];
569                 ac = cpnext->angle - cp->angle;
570                 if(ac < 0.0) ac += 2 * m_pi;
571                 if(ac > maxang) {
572                     maxang = ac;
573                     imaxloop = ic;
574                 }
575             }
576             icend = imaxloop;
577             icstart = imaxloop + 1;
578             if(icstart >= lp->nconnection) icstart = 0;
579             cp = lp->connections[icend];
580             cp->broken = true;
581             done = true;
582         }
583     } while(!done);
584     done_all_connections = false;
585     icstart1 = icstart;
586     //if (debug) printf("Icstart1 = %d\n",icstart1);
587     while(!done_all_connections) {
588         count = 0;
589         done = false;
590         icend = icstart;
591         rooted = false;
592         while(!done) {
593             cp = lp->connections[icend];
```

```
            if(icend == icroot) rooted = true;
            j = icend + 1;
            if(j >= lp->nconnection) {
                j = 0;
            }
            cpnext = lp->connections[j];
            if(connected_connection(cp,cpnext)) {
                if(++count >= lp->nconnection)
                    break;
                icend = j;
            }
            else {
                done = true;
            }
        }
        icmiddle = find_ic_middle(icstart,icend,anchor_connection,acp,lp);
        ic = icup = icdown = icmiddle;
        //if (debug)
        //  printf("IC start = %d  middle = %d  end = %d\n", icstart,icmiddle,icend);
        done = false;
        direction = 0;
        while (!done) {
            if(direction < 0) {
                ic = icup;
            }
            else if(direction == 0) {
                ic = icmiddle;
            }
            else {
                ic = icdown;
            }
            if(ic >= 0) {
                cp = lp->connections[ic];
                if(anchor_connection == NULL || acp != cp) {
                    if(direction == 0) {
                        astart = cp->angle - asin(1.0f / 2.0f / radius);
                        aend = cp->angle + asin(1.0f / 2.0f / radius);
                        bases[cp->start].x = xc + radius * cos(astart);
                        bases[cp->start].y = yc + radius * sin(astart);
                        bases[cp->end].x = xc + radius * cos(aend);
                        bases[cp->end].y = yc + radius * sin(aend);
                    }
                    else if(direction < 0) {
                        j = ic + 1;
                        if(j >= lp->nconnection)
                            j = 0;
                        cp = lp->connections[ic];
                        cpnext = lp->connections[j];
                        cpx = cp->xrad;
                        cpy = cp->yrad;
                        ac = (cp->angle + cpnext->angle) / 2.0f;
                        if(cp->angle > cpnext->angle)
                            ac -= m_pi;
                        cnx = cos(ac);
                        cny = sin(ac);
                        lnx = cny;
                        lny = -cnx;
                        da = cpnext->angle - cp->angle;
                        if(da < 0.0)
                            da += 2 * m_pi;
                        if(cp->extruded) {
                            if(da <= m_pi / 2)
                                rl = 2.0;
                            else
                                rl = 1.5;
                        }
                        else {
```

```
661                             rl = 1.0;
662                         }
663                         bases[cp->end].x = bases[cpnext->start].x + rl*lnx;
664                         bases[cp->end].y = bases[cpnext->start].y + rl*lny;
665                         bases[cp->start].x = bases[cp->end].x + cpy;
666                         bases[cp->start].y = bases[cp->end].y - cpx;
667                     }
668                     else {
669                         j = ic - 1;
670                         if(j < 0)
671                             j = lp->nconnection - 1;
672                         cp = lp->connections[j];
673                         cpnext = lp->connections[ic];
674                         cpnextx = cpnext->xrad;
675                         cpnexty = cpnext->yrad;
676                         ac = (cp->angle + cpnext->angle) / 2.0f;
677                         if(cp->angle > cpnext->angle)
678                             ac -= m_pi;
679                         cnx = cos(ac);
680                         cny = sin(ac);
681                         lnx = -cny;
682                         lny = cnx;
683                         da = cpnext->angle - cp->angle;
684                         if(da < 0.0)
685                             da += 2 * m_pi;
686                         if(cp->extruded) {
687                             if(da <= m_pi / 2)
688                                 rl = 2.0;
689                             else
690                                 rl = 1.5;
691                         }
692                         else {
693                             rl = 1.0;
694                         }
695                         bases[cpnext->start].x = bases[cp->end].x + rl * lnx;
696                         bases[cpnext->start].y = bases[cp->end].y + rl * lny;
697                         bases[cpnext->end].x = bases[cpnext->start].x - cpnexty;
698                         bases[cpnext->end].y = bases[cpnext->start].y + cpnextx;
699                     }
700                 }
701             }
702             if(direction < 0) {
703                 if(icdown == icend) {
704                     icdown = -1;
705                 }
706                 else
707                     if(icdown >= 0) {
708                         if(++icdown >= lp->nconnection) {
709                             icdown = 0;
710                         }
711                     }
712                 direction = 1;
713             }
714             else {
715                 if(icup == icstart)
716                     icup = -1;
717                 else
718                     if(icup >= 0) {
719                         if(--icup < 0) {
720                             icup = lp->nconnection - 1;
721                         }
722                     }
723                 direction = -1;
724             }
725             done = icup == -1 && icdown == -1;
726         }
727         icnext = icend + 1;
```

```
728         if(icnext >= lp->nconnection)
729             icnext = 0;
730         if(icend != icstart && (! (icstart == icstart1 && icnext == icstart1))) {
731             /*
732             *       Move the bases just constructed (or the radius) so
733             *       that the bisector of the end points is radius distance
734             *       away from the loop center.
735             */
736             cp = lp->connections[icstart];
737             cpnext = lp->connections[icend];
738             dx = bases[cpnext->end].x - bases[cp->start].x;
739             dy = bases[cpnext->end].y - bases[cp->start].y;
740             midx = bases[cp->start].x + dx / 2.0f;
741             midy = bases[cp->start].y + dy / 2.0f;
742             rr = sqrt(dx * dx + dy * dy);
743             mx = dx / rr;
744             my = dy / rr;
745             vx = xc - midx;
746             vy = yc - midy;
747             rr = sqrt(dx * dx + dy * dy);
748             vx /= rr;
749             vy /= rr;
750             dotmv = vx * mx + vy * my;
751             nrx = dotmv * mx - vx;
752             nry = dotmv * my - vy;
753             rr = sqrt(nrx * nrx + nry * nry);
754             nrx /= rr;
755             nry /= rr;
756 
757             // Determine which side of the bisector the center should be.
758             dx = bases[cp->start].x - xc;
759             dy = bases[cp->start].y - yc;
760             ac = atan2(dy, dx);
761             if(ac < 0.0)
762                 ac += 2 * m_pi;
763             dx = bases[cpnext->end].x - xc;
764             dy = bases[cpnext->end].y - yc;
765             acn = atan2(dy,dx);
766             if(acn < 0.0)
767                 acn += 2 * m_pi;
768             if(acn < ac)
769                 acn += 2 * m_pi;
770             if(acn - ac > m_pi)
771                 sign = -1;
772             else
773                 sign = 1;
774             nmidx = xc + sign * radius * nrx;
775             nmidy = yc + sign * radius * nry;
776             if (rooted) {
777                 xc -= nmidx - midx;
778                 yc -= nmidy - midy;
779             }
780             else {
781                 for(ic=icstart; ; ++ic >= lp->nconnection ? (ic = 0) : 0) {
782                     cp = lp->connections[ic];
783                     i = cp->start;
784                     bases[i].x += nmidx - midx;
785                     bases[i].y += nmidy - midy;
786                     i = cp->end;
787                     bases[i].x += nmidx - midx;
788                     bases[i].y += nmidy - midy;
789                     if(ic == icend)
790                         break;
791                 }
792             }
793         }
794         icstart = icnext;
```

```
795         done_all_connections = icstart == icstart1;
796     }
797     for(ic = 0; ic < lp->nconnection; ic++) {
798         cp = lp->connections[ic];
799         j = ic + 1;
800         if(j >= lp->nconnection)
801             j = 0;
802         cpnext = lp->connections[j];
803         dx = bases[cp->end].x - xc;
804         dy = bases[cp->end].y - yc;
805         rc = sqrt(dx * dx + dy * dy);
806         ac = atan2(dy, dx);
807         if(ac < 0.0)
808             ac += 2 * m_pi;
809         dx = bases[cpnext->start].x - xc;
810         dy = bases[cpnext->start].y - yc;
811         rcn = sqrt(dx * dx + dy * dy);
812         acn = atan2(dy, dx);
813         if(acn < 0.0)
814             acn += 2 * m_pi;
815         if(acn < ac)
816             acn += 2 * m_pi;
817         dan = acn - ac;
818         dcp = cpnext->angle - cp->angle;
819         if(dcp <= 0.0)
820             dcp += 2 * m_pi;
821         if(fabs(dan - dcp) > m_pi) {
822             if(cp->extruded) {
823                 //printf("Warning from traverse_loop. Loop %d has crossed regions\n",↙
     lp->number);
824             }
825             else {
826                 cp->extruded = true;
827                 goto set_radius; // Forever shamed
828             }
829         }
830         if (cp->extruded) {
831             construct_extruded_segment(cp, cpnext);
832         }
833         else {
834             n = cpnext->start - cp->end;
835             if(n < 0)
836                 n += nbase + 1;
837             angleinc = dan / n;
838             for(j = 1;  j < n;  j++) {
839                 i = cp->end + j;
840                 if(i > nbase)
841                     i -= nbase + 1;
842                 a = ac + j * angleinc;
843                 rr = rc + (rcn - rc) * (a - ac) / dan;
844                 bases[i].x = xc + rr * cos(a);
845                 bases[i].y = yc + rr * sin(a);
846             }
847         }
848     }
849     for(ic=0; ic < lp->nconnection; ic++) {
850         if(icroot != ic) {
851             cp = lp->connections[ic];
852             generate_region(cp);
853             traverse_loop(cp->loop, cp);
854         }
855     }
856     n = 0;
857     sx = 0.0;
858     sy = 0.0;
859     for(ic = 0;  ic < lp->nconnection;  ic++) {
860         j = ic + 1;
```

```
861         if(j >= lp->nconnection)
862             j = 0;
863         cp = lp->connections[ic];
864         cpnext = lp->connections[j];
865         n += 2;
866         sx += bases[cp->start].x + bases[cp->end].x;
867         sy += bases[cp->start].y + bases[cp->end].y;
868         if(!cp->extruded) {
869             for(j = cp->end + 1;  j != cpnext->start;  j++) {
870                 if(j > nbase)
871                     j -= nbase + 1;
872                 n++;
873                 sx += bases[j].x;
874                 sy += bases[j].y;
875             }
876         }
877     }
878     lp->x = sx / n;
879     lp->y = sy / n;
880 }
881
882 // FUNCTION:  determine_radius
883 // For the loop pointed to by lp, determine the radius of
884 // the loop that will ensure that each base around the loop will
885 // have a separation of at least minbaseseparation around the circle.
886 // If a segment joining two connectors will not support this separation,
887 // then the flag, extruded, will be set in the first of these
888 // two indicators. The radius is set in lp.
889 //
890 // The radius is selected by a least squares procedure where the sum of the
891 // squares of the deviations of length from the ideal value of 1 is used
892 // as the error function.
893 //
894 // @param   lp                  struct Loop*    a pointer to the loop
895 // @param   minbaseseparation   float           the mimimum base separaton
896 // @return  void
897 void RNAGraph::determine_radius(struct Loop *lp, float minbaseseparation) {
898     float mindit, ci, dt, sumn, sumd, radius, dit;
899     int count, i, j, end, start, imindit;
900     struct Connection *cp, *cpnext;
901     float rt2_2 = 0.7071068f;
902
903     count = 0;
904     do {
905         mindit = 1.0e10;
906         for (sumd=0.0, sumn=0.0, i=0; i < lp->nconnection; i++) {
907             cp = lp->connections[i];
908             j = i + 1;
909             if (j >= lp->nconnection)
910                 j = 0;
911             cpnext = lp->connections[j];
912             end =  cp->end;
913             start = cpnext->start;
914             if (start < end)
915                 start += nbase + 1;
916             dt = cpnext->angle - cp->angle;
917             if (dt <= 0.0)
918                 dt += 2 * m_pi;
919             if (!cp->extruded)
920                 ci = start - end;
921             else {
922                 if (dt <= m_pi / 2)
923                     ci = 2.0;
924                 else
925                     ci = 1.5;
926             }
927             sumn += dt * (1.0f / ci + 1.0f);
```

```
928             sumd += dt * dt / ci;
929             dit = dt / ci;
930             if (dit < mindit && !cp->extruded && ci > 1.0) {
931                 mindit = dit;
932                 imindit = i;
933             }
934         }
935         radius = sumn / sumd;
936         if (radius < rt2_2)
937             radius = rt2_2;
938         if (mindit * radius < minbaseseparation) {
939             lp->connections[imindit]->extruded = true;
940         }
941     } while (mindit * radius < minbaseseparation);
942 
943     if (lp->radius > 0.0)
944         radius = lp->radius;
945     else
946         lp->radius = radius;
947 }
948 
949 // FUNCTION:  connected_connection
950 // Determines if the connections cp and cpnext are connected
951 //
952 // @param   cp      struct Connection*   the pointer to the connection
953 // @param   cpnext  struct Connection*   the pointer to the next connection
954 // @return  bool
955 bool RNAGraph::connected_connection(struct Connection *cp, struct Connection *cpnext)↙
     {
956 
957     if (cp->extruded) {
958         return true;
959     }
960     else if (cp->end+1 == cpnext->start) {
961         return true;
962     }
963     else {
964         return false;
965     }
966 }
967 
968 // FUNCTION:  find_ic_middle
969 // Finds the middle of a set of connected connectors. This is normally
970 // the middle connection in the sequence except if one of the connections
971 // is the anchor, in which case that connection will be used.
972 //
973 // @param   icstart             int                  the icstart position
974 // @param   icent               int                  the icent position
975 // @param   anchor_connection   struct Connection*   the anchor connection pointer
976 // @param   acp                 struct Connection*   pointer to connection
977 // @param   lp                  struct Loop*         pointer to loop
978 // @return  int
979 int RNAGraph::find_ic_middle(int icstart, int icend, struct Connection *            ↙
    anchor_connection, struct Connection *acp, struct Loop *lp) {
980     int count, ret, ic, i;
981     bool done;
982 
983     count = 0;
984     ret = -1;
985     ic = icstart;
986     done = false;
987     while (!done) {
988         if (count++ > lp->nconnection * 2) {
989             throw "RNAGraph: Infinite loop detected in find_ic_middle";
990         }
991         if (anchor_connection != NULL && lp->connections[ic] == acp) {
992             ret = ic;
```

```
 993         }
 994         done = ic == icend;
 995         if (++ic >= lp->nconnection) {
 996             ic = 0;
 997         }
 998     }
 999     if (ret == -1) {
1000         for (i=1, ic=icstart; i<(count+1)/2; i++) {
1001             if (++ic >= lp->nconnection) ic = 0;
1002         }
1003         ret = ic;
1004     }
1005     return ret;
1006 }
1007
1008 // FUNCTION:  generate_region
1009 // Generates the coordinates for the base pairing region of a connection
1010 // given the position of the starting base pair.
1011 //
1012 // @param   cp   struct Connection*   pointer to Connection structure
1013 // @return  void
1014 void RNAGraph::generate_region(struct Connection *cp) {
1015     int l, start, end, i, mate;
1016     struct Region *rp;
1017
1018     rp = cp->region;
1019     l = 0;
1020     if (cp->start == rp->start1) {
1021         start = rp->start1;
1022         end = rp->end1;
1023     }
1024     else {
1025         start = rp->start2;
1026         end = rp->end2;
1027     }
1028     if (bases[cp->start].x > m_anum - 100.0 || bases[cp->end].x > m_anum - 100.0) {
1029         throw "RNAGraph: Bad region passed to generate_region. Coordinates not
     defined";
1030     }
1031     for (i=start+1; i<=end; i++) {
1032         l++;
1033         bases[i].x = bases[cp->start].x + l * cp->xrad;
1034         bases[i].y = bases[cp->start].y + l * cp->yrad;
1035         mate = bases[i].mate;
1036         bases[mate].x = bases[cp->end].x + l * cp->xrad;
1037         bases[mate].y = bases[cp->end].y + l * cp->yrad;
1038     }
1039 }
1040
1041 // FUNCTION:  construct_circle_segment
1042 // Draws the segment of residue between the bases numbered start
1043 // through end, where start and end are presumed to be part of a base
1044 // pairing region. They are drawn as a circle which has a chord given
1045 // by the ends of two base pairing regions defined by the connections.
1046 //
1047 // @param   start   int   the start of the circle segment
1048 // @param   end     int   the end of the circle segment
1049 // @return  void
1050 void RNAGraph::construct_circle_segment(int start, int end) {
1051     float dx, dy, rr, h, angleinc, midx, midy, xn, yn, nrx, nry, mx, my, a;
1052     int l, j, i;
1053
1054     dx = bases[end].x - bases[start].x;
1055     dy = bases[end].y - bases[start].y;
1056     rr = sqrt(dx*dx + dy*dy);
1057     l = end - start;
1058     if (l < 0)
```

```
1059            l += nbase + 1;
1060        if (rr >= l) {
1061            dx /= rr;
1062            dy /= rr;
1063            for (j = 1;  j < l;  j++) {
1064                i = start + j;
1065                if (i > nbase)
1066                    i -= nbase + 1;
1067                bases[i].x = bases[start].x + dx * (float)j / (float)l;
1068                bases[i].y = bases[start].y + dy * (float)j / (float)l;
1069            }
1070        }
1071        else {
1072            find_center_for_arc(l-1, rr, &h, &angleinc);
1073            dx /= rr;
1074            dy /= rr;
1075            midx = bases[start].x + dx * rr / 2.0f;
1076            midy = bases[start].y + dy * rr / 2.0f;
1077            xn = dy;
1078            yn = -dx;
1079            nrx = midx + h * xn;
1080            nry = midy + h * yn;
1081            mx = bases[start].x - nrx;
1082            my = bases[start].y - nry;
1083            rr = sqrt(mx * mx + my * my);
1084            a = atan2(my, mx);
1085            for (j = 1;  j < l;  j++) {
1086                i = start + j;
1087                if (i > nbase)
1088                    i -= nbase + 1;
1089                bases[i].x = nrx + rr * cos(a + j * angleinc);
1090                bases[i].y = nry + rr * sin( a + j * angleinc);
1091            }
1092        }
1093    }
1094
1095    // FUNCTION:  construct_extruded_segment
1096    // Constructs the segment between cp and cpnext as a circle if possible.
1097    // However, if the segment is too large, the lines are drawn between
1098    // the two connecting regions, and bases are placed there until the
1099    // connecting circle will fit.
1100    //
1101    // @param   cp       struct Connection*   pointer to a Connection structure
1102    // @param   cpnext   struct Connection*   pointer to a Connection structure
1103    // @return  void
1104    void RNAGraph::construct_extruded_segment(struct Connection *cp, struct Connection * ↙
        cpnext) {
1105        float astart, aend1, aend2, aave, dx, dy, a1, a2, ac, rr, da, dac;
1106        int start, end, n, nstart, nend;
1107        bool collision;
1108
1109        astart = cp->angle;
1110        aend2 = aend1 = cpnext->angle;
1111        if (aend2 < astart)
1112            aend2 += 2 * m_pi;
1113        aave = (astart + aend2) / 2.0f;
1114        start = cp->end;
1115        end = cpnext->start;
1116        n = end - start;
1117        if (n < 0)
1118            n += nbase + 1;
1119        da = cpnext->angle - cp->angle;
1120        if (da < 0.0) {
1121            da += 2 * m_pi;
1122        }
1123        if (n == 2)
1124            construct_circle_segment(start, end);
```

```
1125        else {
1126            dx = bases[end].x - bases[start].x;
1127            dy = bases[end].y - bases[start].y;
1128            rr = sqrt(dx * dx + dy * dy);
1129            dx /= rr;
1130            dy /= rr;
1131            if (rr >= 1.5 && da <= m_pi / 2) {
1132                nstart = start + 1;
1133                if (nstart > nbase)
1134                    nstart -= nbase + 1;
1135                nend = end - 1;
1136                if (nend < 0)
1137                    nend += nbase + 1;
1138                bases[nstart].x = bases[start].x + 0.5f * dx;
1139                bases[nstart].y = bases[start].y + 0.5f * dy;
1140                bases[nend].x = bases[end].x - 0.5f * dx;
1141                bases[nend].y = bases[end].y - 0.5f * dy;
1142                start = nstart;
1143                end = nend;
1144            }
1145            do {
1146                collision = false;
1147                construct_circle_segment(start, end);
1148                nstart = start + 1;
1149                if (nstart > nbase)
1150                    nstart -= nbase + 1;
1151                dx = bases[nstart].x - bases[start].x;
1152                dy = bases[nstart].y - bases[start].y;
1153                a1 = atan2(dy, dx);
1154                if (a1 < 0.0)
1155                    a1 += 2 * m_pi;
1156                dac = a1 - astart;
1157                if (dac < 0.0)
1158                    dac += 2 * m_pi;
1159                if (dac > m_pi)
1160                    collision = true;
1161                nend = end - 1;
1162                if (nend < 0)
1163                    nend += nbase + 1;
1164                dx = bases[nend].x - bases[end].x;
1165                dy = bases[nend].y - bases[end].y;
1166                a2 = atan2(dy, dx);
1167                if (a2 < 0.0)
1168                    a2 += 2 * m_pi;
1169                dac = aend1 - a2;
1170                if (dac < 0.0)
1171                    dac += 2 * m_pi;
1172                if (dac > m_pi)
1173                    collision = true;
1174                if (collision) {
1175                    ac = minf2(aave, astart + 0.5f);
1176                    bases[nstart].x = bases[start].x + cos(ac);
1177                    bases[nstart].y = bases[start].y + sin(ac);
1178                    start = nstart;
1179                    ac = maxf2(aave, aend2 - 0.5f);
1180                    bases[nend].x = bases[end].x + cos(ac);
1181                    bases[nend].y = bases[end].y + sin(ac);
1182                    end = nend;
1183                    n -= 2;
1184                }
1185            } while (collision && n > 1);
1186        }
1187    }
1188
1189    // FUNCTION:  find_center_for_arc
1190    // Given n points to be placed equidistantly and equiangularly on a
1191    // polygon which has a chord of length, b, find the distance, h, from the
```

```
1192 // midpoint of the chord for the center of polygon. Positive values
1193 // mean the center is within the polygon and the chord, whereas
1194 // negative values mean the center is outside the chord. Also, the
1195 // radial angle for each polygon side is returned in theta.
1196 //
1197 // The procedure uses a bisection algorithm to find the correct
1198 // value for the center. Two equations are solved, the angles
1199 // around the center must add to 2*pi, and the sides of the polygon
1200 // excluding the chord must have a length of 1.
1201 //
1202 // @param   n        int
1203 // @param   b        float
1204 // @param   hp       float*
1205 // @param   thetap   float*
1206 // @return  void
1207 void RNAGraph::find_center_for_arc(int n, float b, float *hp, float *thetap) {
1208     float h, hhi, hlow, r, disc, theta, e, phi;
1209     int iter;
1210
1211     hhi = (n + 1) / m_pi;
1212     hlow = - hhi - b / (n + 1 - b);
1213     iter = 0;
1214     do {
1215         h = (hhi + hlow) / 2.0f;
1216         r = sqrt(h * h + b * b / 4.0f);
1217         disc = 1.0f - 1.0f / 2.0f / (r * r);
1218         if (fabs(disc) > 1.0) {
1219             throw "RNAGraph: Unexpected large magnitude discriminant = %g"; // disc;
1220         }
1221         theta = acos(disc);
1222         phi = acos(h / r);
1223         e = theta * (n + 1) + 2 * phi - 2 * m_pi;
1224         if (e > 0.0) {
1225             hlow = h;
1226         }
1227         else {
1228             hhi = h;
1229         }
1230     } while (fabs(e) > 0.0001 && ++iter < maxiter);
1231     if (iter >= maxiter) {
1232         //printf("Iteration failed in find_center_for_arc\n");
1233         h = 0.0;
1234         theta = 0.0;
1235     }
1236     *hp = h;
1237     *thetap = theta;
1238 }
1239
1240 // FUNCTION:  Write_PLT2_Output
1241 // Writes the coordinates as a PLT2 command stream
1242 //
1243 // @param   dev          char*   PLT2 device string
1244 // @param   outputfile   char*   The file name of the PLT2 file to create
1245 // @param   xsize        float   The x coordinate size
1246 // @param   ysize        float   the y coordinate size
1247 // @param   mark_loops   bool    flag to determine if loop numbers should be drawn
1248 // @param   draw_bases   bool    flag to determine if the base symbol should be drawn
1249 // @param   csz          float   the character size
1250 // @param   label_rate   int     the rate a which to label the bases
1251 // @param   dot_pairs    bool    flag to determine if a dot or line should be used
1252 // @param   mosaicx      int     number of frames in mosaic x
1253 // @param   mosaicy      int     number of frames in mosaic y
1254 // @param   glob_rot     float   global rotation angle
1255 // @return  void
1256 void RNAGraph::Write_PLT2_Output(char *dev, char *outputfile, float xsize, float ↙
        ysize, bool mark_loops, bool draw_bases, float csz, int label_rate, bool ↙
        dot_pairs, int mosaicx, int mosaicy, float glob_rot) {
```

```
1257     int i, mate, imx, imy, pagecnt, istart;
1258     float xmin, xmax, ymin, ymax, scalex, scaley, scale, scalecsz;
1259     float xmn, xmx, ymn, ymx;
1260     float xn, yn, r, x1, y1, x2, y2, xs, ys, xoff, yoff, ct, st, xr, yr;
1261     struct Loop *lp;
1262 
1263     #define okx(x) (xmn - scale <= (x) && (x) <= xmx + scale)
1264     #define oky(y) (ymn - scale <= (y) && (y) <= ymx + scale)
1265 
1266     FILE *outf;
1267     outf = fopen(outputfile,"w");
1268     if (outf == NULL) {
1269         throw "RNAGraph: Unable to open output file";
1270     }
1271 
1272     if(label_rate < 0) // Make sure label_rate is valid
1273         label_rate = 0;
1274 
1275     if(mosaicx < 1) // Make sure mosaic is valid
1276         mosaicx = 1;
1277     if(mosaicy < 1)
1278         mosaicy = 1;
1279 
1280     fprintf(outf,"DEV %s\n", dev);
1281     fprintf(outf,"CSZ %10.4f\n", csz);
1282     ct = cos(glob_rot * m_pi / 180.0f);
1283     st = sin(glob_rot * m_pi / 180.0f);
1284     for (i = 0; i <= nbase; i++) {
1285         xr = bases[i].x * ct + bases[i].y * st;
1286         yr = bases[i].y * ct - bases[i].x * st;
1287         bases[i].x = xr;
1288         bases[i].y = yr;
1289     }
1290     for (i = 0; i < loop_count; i++) {
1291         lp = loops + i;
1292         xr = lp->x * ct + lp->y * st;
1293         yr = lp->y * ct - lp->x * st;
1294         lp->x = xr;
1295         lp->y = yr;
1296     }
1297     xmax = ymax = -m_bignum;
1298     xmin = ymin = m_bignum;
1299     for (i = 0; i <= nbase; i++) {
1300         if (bases[i].x > m_anum - 100.0) {
1301             //printf("\nError in write_plt2_output -- base %d position is undefined.\↙
     n",i);
1302         }
1303         else {
1304             xmax = maxf2(xmax,bases[i].x);
1305             ymax = maxf2(ymax,bases[i].y);
1306             xmin = minf2(xmin,bases[i].x);
1307             ymin = minf2(ymin,bases[i].y);
1308         }
1309     }
1310     scalex = (xmax - xmin) / xsize / (float) mosaicx;
1311     scaley = (ymax - ymin) / ysize / (float) mosaicy;
1312     scale = maxf2(scalex, scaley) * 1.02f;
1313     /* The line below was added by Darrin Stewart $$$$$$$$*/
1314     csz = csz / scale;
1315     /* It works best when csz is .3, but numbers from .4 to .1 are reasonable*/
1316     /* The character size is now depended on the internal scaling of the image*/
1317     /* This seems to be quite accurate. */
1318     xs = xsize * scale;
1319     ys = ysize * scale;
1320     xoff = (xs * mosaicx - xmax + xmin) / 2.0f;
1321     yoff = (ys * mosaicy - ymax + ymin) / 2.0f;
1322     fprintf(outf, "SA %g\n", scale);
```

```
1323        fprintf(outf, "ORI 0.0 0.0\n");
1324        pagecnt = 0;
1325        for (imx = 1; imx <= mosaicx; imx++) {
1326            for (imy = 1; imy <= mosaicy; imy++) {
1327                if (++pagecnt > 1) {
1328                    fprintf(outf, "DUMP\n");
1329                }
1330                if (title[0] != '\0') {
1331                    if (imx == (mosaicx + 1) / 2 && imy == 1) {
1332                        fprintf(outf, "CSZ 0.4\n");
1333                        fprintf(outf, "CTA %10.3f 1.0 \"%s\"\n", xsize / 2.0, title);
1334                        fprintf(outf, "CSZ %10.4f\n", csz);
1335                    }
1336                }
1337                xmn = xmin + (imx - 1) * xs - xoff;
1338                ymn = ymin + (imy - 1) * ys - yoff;
1339                xmx = xmin + imx * xs - xoff;
1340                ymx = ymin + imy * ys - yoff;
1341                fprintf(outf, "OD %10.3f %10.3f\n", xmn, ymn);
1342                fprintf(outf, "COLOR WHITE\n");
1343                if (bases[0].x != m_anum)
1344                    fprintf(outf,"BRI 2\nMOV %10.3f %10.3f\nCIA 0.05\nCIA 0.1\nCIA 0.15\n↙
    ", bases[0].x, -bases[0].y);
1345                fprintf(outf, "BRI 3\n");
1346                if (draw_bases) {
1347                    fprintf(outf, "CM BASES %d\n", nbase);   /* Zuker adds nbase */
1348                    for (i = 1; i <= nbase; i++) {
1349                        if (bases[i].x != m_anum)
1350                            if (okx(bases[i].x) && oky(bases[i].y))
1351                                fprintf(outf, "CTX %10.3f %10.3f \"%c\"\n", bases[i].x, -↙
    bases[i].y, bases[i].name);
1352                    }
1353                }
1354                fprintf(outf, "CM SEQUENCE LINES\n");
1355                if (draw_bases)
1356                    scalecsz = scale * csz * 1.8f;
1357                else
1358                    scalecsz = 0.0;
1359                for (i = 0; i <= nbase-1; i++) {
1360                    if (bases[i].x != m_anum && bases[i+1].x != m_anum) {
1361                        if (okx(bases[i].x) && oky(bases[i].y) || okx(bases[i+1].x) &&   ↙
    oky(bases[i+1].y)) {
1362                            xn = bases[i+1].x - bases[i].x;
1363                            yn = bases[i+1].y - bases[i].y;
1364                            r = sqrt(xn * xn + yn * yn);
1365                            if (r > scalecsz) {
1366                                xn /= r;
1367                                yn /= r;
1368                                x1 = bases[i].x + xn * scalecsz / 2.0f;
1369                                y1 = bases[i].y + yn * scalecsz / 2.0f;
1370                                x2 = bases[i+1].x - xn * scalecsz / 2.0f;
1371                                y2 = bases[i+1].y - yn * scalecsz / 2.0f;
1372                                fprintf(outf,"LI %10.3f %10.3f 0.0 %10.3f %10.3f 0.0\n", ↙
    x1, -y1, x2, -y2);
1373                            }
1374                        }
1375                    }
1376                }
1377                fprintf(outf, "CM BASE PAIRING LINES WITH BASE PAIRS\n");
1378                fprintf(outf, "COLOR RED\nBRI 5\n");
1379
1380                #define draw_1bp \
1381                if (dot_pairs) {\
1382                    x1 = (bases[i].x + bases[mate].x) / 2.0f;\
1383                    y1 = (bases[i].y + bases[mate].y) / 2.0f;\
1384                    if (okx(x1) && oky(y1)) \
1385                        fprintf(outf,"MOV %10.3f %10.3f %5d %5d\nCIA 0.0\n",x1,-y1,i,     ↙
```

```
    mate);\
1386            }\
1387            else {\
1388                xn = bases[mate].x - bases[i].x;\
1389                yn = bases[mate].y - bases[i].y;\
1390                r = sqrt(xn*xn + yn*yn);\
1391                if (r > scalecsz) {\
1392                    xn /= r;\
1393                    yn /= r;\
1394                    x1 = bases[i].x + xn * scalecsz / 2.0f;\
1395                    y1 = bases[i].y + yn * scalecsz / 2.0f;\
1396                    x2 = bases[mate].x - xn * scalecsz / 2.0f;\
1397                    y2 = bases[mate].y - yn * scalecsz / 2.0f;\
1398                    if (okx(x1) && oky(y1) || okx(x2) && oky(y2)) \
1399                        fprintf(outf,"LI %10.3f %10.3f 0.0 %10.3f %10.3f 0.0 %5d %5d\↙
    n", x1, -y1, x2, -y2, i, mate);\
1400                }\
1401            }
1402
1403            for (i = 0; i <= nbase; i++) {
1404                if ((mate = bases[i].mate) && i < mate ) {
1405                    if (bases[i].x != m_anum && bases[mate].x != m_anum) {
1406                        if (bases[i].name == 'G' && bases[mate].name == 'C' || bases ↙
    [i].name == 'C' && bases[mate].name == 'G') {
1407                            draw_1bp;
1408                        }
1409                    }
1410                }
1411            }
1412
1413            fprintf(outf, "COLOR MAGENTA\nBRI 1\n");
1414            for (i = 0; i <= nbase; i++) {
1415                if ((mate = bases[i].mate) && i < mate ) {
1416                    if (bases[i].x != m_anum && bases[mate].x != m_anum) {
1417                        if (!(bases[i].name == 'G' && bases[mate].name == 'C' ||      ↙
    bases[i].name == 'C' && bases[mate].name == 'G')) {
1418                            draw_1bp;
1419                        }
1420                    }
1421                }
1422            }
1423
1424            fprintf(outf, "COLOR WHITE\nBRI 3\n");
1425            if (label_rate > 0) {
1426                fprintf(outf,"CM LABELS %d\nCSZ %10.4f\n", nbase, csz);
1427                istart = label_rate*(1 + bases[1].hist_num / label_rate) - bases[1]. ↙
    hist_num + 1;
1428                for (i = istart; i <= nbase; i += label_rate) {
1429                    if (bases[i].x != m_anum) {
1430                        if (okx(bases[i].x) && oky(bases[i].y)) {
1431                            float dx, dy, angle;
1432                            int ia;
1433                            if (i == nbase) {
1434                                dx = bases[i].x - bases[i-1].x;
1435                                dy = bases[i].y - bases[i-1].y;
1436                            }
1437                            else {
1438                                dx = bases[i+1].x - bases[i].x;
1439                                dy = bases[i+1].y - bases[i].y;
1440                            }
1441                            angle = atan2(dy, dx) / m_dtor - 90.0f;
1442                            ia = angle;
1443                            fprintf(outf, "TEX %d %10.3f %10.3f \"  %d\"\n", ia,      ↙
    bases[i].x, -bases[i].y, bases[i].hist_num);
1444                        }
1445                    }
1446                }
```

```
1447             }
1448
1449             if (mark_loops) {
1450                 fprintf(outf,"CM LOOP LABELS\n");
1451                 for (i = 0; i < loop_count; i++) {
1452                     lp = loops + i;
1453                     if (okx(lp->x) && oky(lp->y))
1454                         fprintf(outf, "CTX %10.3f %10.3f \"%d\"\n", lp->x, -lp->y, lp↙
     ->number);
1455                 }
1456             }
1457         }
1458     }
1459 }
1460
1461 // FUNCTION:  minf2
1462 // Computes the minimum of two floating point numbers
1463 //
1464 // @param   x1   float   the first float
1465 // @param   x2   float   the second float
1466 // @return  float
1467 float RNAGraph::minf2(float x1, float x2) {
1468     return x1 < x2 ? x1 : x2;
1469 }
1470
1471 // FUNCTION:  maxf2
1472 // Computes the maximum of two floating point numbers
1473 //
1474 // @param   x1    float   the first float
1475 // @param   x2    float   the second float
1476 // @return  float
1477 float RNAGraph::maxf2(float x1, float x2) {
1478     return x1 > x2 ? x1 : x2;
1479 }
1480
1481 // FUNCTION:  trim
1482 // Trims blanks off the end of st, *stlenp gives the current length of st
1483 //
1484 // @param   st      char*   the character array to trim
1485 // @param   stlenp  int*    the length of the character array
1486 // @return  void
1487 void RNAGraph::trim(char *st, int *stlenp) {
1488     int stlen = *stlenp;
1489
1490     while (stlen > 0) {
1491         if (st[stlen-1] != ' ') break;
1492         stlen -= 1;
1493     }
1494     *stlenp = stlen;
1495 }
1496
1497 // FUNCTION:  chcnbl
1498 // Converts all non-printable control characters into blanks
1499 //
1500 // @param   st    char*   the character array to clear characters from
1501 // @param   stlen int*    the length of the character array
1502 // @return  void
1503 void RNAGraph::chcnbl(char *st, int stlen) {
1504     int i;
1505
1506     for (i = 1; i <= stlen; i++) {
1507         if (*st < 32 || *st > 126)
1508             *st = ' ';
1509         st += 1;
1510     }
1511 }
1512
```

```
1513 // FUNCTION:  setMinBaseSeparation
1514 // Used to specify the minimum permissible separation between bases (Ex. 1.0)
1515 //
1516 // @param   MinBaseSeparation   float   the minimum base separation
1517 // @return  void
1518 void RNAGraph::SetMinBaseSeparation(float MinBaseSeparation) {
1519     //printf("A size > 1.0 will be reduced to 1.0.\n");
1520     MinBaseSeparation = maxf2(0.0, minf2(1.0, MinBaseSeparation));
1521     m_minbaseseparation = MinBaseSeparation;
1522 }
1523
1524 // FUNCTION:  GetMinBaseSeparation
1525 // Used to retrieve the minimum base separation
1526 //
1527 // @param   undefined   void
1528 // @return  float
1529 float RNAGraph::GetMinBaseSeparation(void) {
1530     return m_minbaseseparation;
1531 }
1532
1533 // FUNCTION:  ChangeLoopRadii
1534 // Used to change the loop radii
1535 //
1536 // @param   undefined   void
1537 // @return  void
1538 void RNAGraph::ChangeLoopRadii() {
1539     bool change_rad;
1540     int ilp;
1541     float r;
1542     struct Radloop *rlp;
1543
1544     rlphead = rlptail = NULL;
1545     //change_rad = ask("Do you want to change loop radii?");
1546     change_rad = false;
1547     if (change_rad) {
1548         do {
1549             //printf("Please specify a loop number and radius. Type zeroes to quit:\n↙
     ");
1550             r = 0.0;
1551             scanf("%d%f",&ilp,&r);
1552             if (ilp > 0) {
1553                 if (r <= 0.0) {
1554                     //printf("Bad radius specified. It must be positive.\n");
1555                 }
1556                 else {
1557                     rlp = struct_alloc(Radloop);
1558                     rlp->radius = r;
1559                     rlp->loopnumber = ilp;
1560
1561                     // Add to double list
1562                     rlp->next = rlp->prev = NULL;
1563                     if(rlphead == NULL)
1564                         rlphead = rlptail = rlp;
1565                     else {
1566                         rlptail->next = rlp;
1567                         rlp->prev = rlptail;
1568                         rlptail = rlp;
1569                     }
1570                 }
1571             }
1572         }
1573         while (ilp > 0);
1574     }
1575 }
1576
1577 // FUNCTION:  GetPi
1578 // Used to retrieve the value of PI
```

```
1579 //
1580 // @param   undefined   void
1581 // @return  float
1582 float RNAGraph::GetPi(void) {
1583     return m_pi;
1584 }
1585
1586 // FUNCTION:  GetBases
1587 // Used to retrieve that base_tag struct
1588 //
1589 // @param   undefined   void
1590 // @return  struct base_tag*
1591 struct RNAGraph::base_tag *RNAGraph::GetBases() {
1592     return bases;
1593 }
1594
1595 // FUNCTION:  GetNBase
1596 // Used to retrieve the number of bases
1597 //
1598 // @param   undefined   void
1599 // @return  int
1600 int RNAGraph::GetNBase(void) {
1601     return nbase;
1602 }
1603
1604 // FUNCTION:  GetLoopCount
1605 // Used to retrieve the number of loops
1606 //
1607 // @param   undefined   void
1608 // @return  int
1609 int RNAGraph::GetLoopCount(void) {
1610     return loop_count;
1611 }
1612
1613 // FUNCTION:  GetLoops
1614 // Used to retrieve the loop_tag structure
1615 //
1616 // @param   undefined   void
1617 // @return  struct loop_tag*
1618 struct RNAGraph::loop_tag *RNAGraph::GetLoops() {
1619     return loops;
1620 }
1621
1622 // FUNCTION:  GetBigNum
1623 // Used to retrieve the Big Number (a place holder number)
1624 //
1625 // @param   undefined   void
1626 // @return  float
1627 float RNAGraph::GetBigNum(void) {
1628     return m_bignum;
1629 }
1630
1631 // FUNCTION:  GetANum
1632 // Used to retrieve the ANum
1633 //
1634 // @param   undefined   void
1635 // @return  float
1636 float RNAGraph::GetANum(void) {
1637     return m_anum;
1638 }
1639
1640 // FUNCTION:  GetTitle
1641 // Used to retrieve the sequence title
1642 //
1643 // @param   undefined   void
1644 // @return  char*
1645 char *RNAGraph::GetTitle(void) {
```

```
1646     return title;
1647 }
1648
1649 // FUNCTION:  GetDTor
1650 //
1651 // @param   undefined   void
1652 // @return  float
1653 float RNAGraph::GetDTor(void) {
1654     return m_dtor;
1655 }
```

```
/*****************************************************
* Copyright (c): 2006, All Rights Reserved
* Project:       SJSU Masters Project
* File:          RNA.h
* Purpose:       Header file for class to store RNA information
*
* Start Date:    8/1/2006
* Programmer:    Brandon Hunter
*
*****************************************************/

#pragma once

class RNA
{
private:
    typedef struct planarPos_tag {
        float PlanarX, PlanarY;
    } PlanarPos_type;

    int Length; // The Length of the RNA
    char* Title; // The title of the RNA
    char* Sequence; // The RNA sequence
    int StepPosition; // 1 based position when stepping through RNA sequence
    int* Pairing; // Array that stores pairing positions
    struct PlanarPos_type* PlanarPos; // Pointer to array of PlanarPos_type (used for drawing lines from Matrix to Planar graph)

public:
    RNA(char* Title, char* Sequence, int Length);
    ~RNA(void);

    char* getTitle(void);
    char* getSequence(void);
    int getLength(void);
    void setStepPosition(int StepPosition);
    int getStepPosition(void);
    void setPairing(int* Pairing);
    int* getPairing(void);
    struct PlanarPos_type* GetPlanarPos();
};

```

```cpp
/*****************************************************
* Copyright (c): 2006, All Rights Reserved
* Project:       SJSU Masters Project
* File:          RNA.cpp
* Purpose:       Class implementation to store RNA information
*
* Start Date:    8/1/2006
* Programmer:    Brandon Hunter
*
*****************************************************/

#include "StdAfx.h"
#include "RNA.h"

// FUNCTION:  RNA
// Default Constructor
//
// @param   Title     char*   the title of the RNA sequence
// @param   Sequence  char*   the sequence of nucleotide bases
// @param   Lenth     int     the length of the RNA sequence
// @return  void
RNA::RNA(char* Title, char* Sequence, int Length) {
    RNA::Title = Title; // Initialize sequence title
    RNA::Sequence = Sequence; // Initialize sequence
    RNA::Length = Length; // Initialize Sequence Length
    StepPosition = Length;  // Initialize StepPosition to full length of RNA sequence
    Pairing = NULL;
    PlanarPos = new PlanarPos_type[Length];
}

// FUNCTION:  ~RNA
// Default Destructor
//
// @param   undefined   void
// @return  void
RNA::~RNA(void) {
    if(Title != NULL)
    {
        delete[] Title;
        Title = NULL;
    }
    if(Sequence != NULL)
    {
        delete[] Sequence;
        Sequence = NULL;
    }
    if(Pairing != NULL)
    {
        delete[] Pairing;
        Pairing = NULL;
    }
    if(PlanarPos != NULL)
    {
        delete[] PlanarPos;
        PlanarPos = NULL;
    }
}

// FUNCTION:  getTitle
// Used to retrieve the RNA Title
//
// @param   undefined   void
// @return  char*
char* RNA::getTitle(void) {
    return Title;
}

```

```
68 // FUNCTION:  getSequence
69 // Used to retrieve the Sequence
70 //
71 // @param   undefined   void
72 // @return  char*
73 char* RNA::getSequence(void) {
74     return Sequence;
75 }
76
77 // FUNCTION:  getLength
78 // Used to Retrieve the length of the RNA sequence
79 //
80 // @param   undefined   void
81 // @return  int
82 int RNA::getLength(void) {
83     return Length;
84 }
85
86 // FUNCTION:  setStepPosition
87 // Used to set the current step position within the sequence
88 //
89 // @param   StepPosition   int   the step position to set
90 // @return  void
91 void RNA::setStepPosition(int StepPosition) {
92     if(StepPosition > Length)
93         throw "Step position is greater than length of sequence";
94
95     RNA::StepPosition = StepPosition;
96 }
97
98 // FUNCTION:  getStepPosition
99 // Used to retrieve the current step position
100 //
101 // @param   undefined   void
102 // @return  int
103 int RNA::getStepPosition(void) {
104     return StepPosition;
105 }
106
107 // FUNCTION:  setPairing
108 // Used to set the Pairing array
109 //
110 // @param   Pairing   int*   the integer array that is the pairing
111 // @return  void
112 void RNA::setPairing(int* Pairing) {
113     if(RNA::Pairing != NULL)
114         delete[] RNA::Pairing;
115
116
117     RNA::Pairing = Pairing;
118 }
119
120 // FUNCTION:  getPairing
121 // Used to retrieve the Pairing array
122 //
123 // @param   undefined   void
124 // @return  int*
125 int* RNA::getPairing(void) {
126     return Pairing;
127 }
128
129 // FUNCTION:  GetPlanarPos
130 // Used to retrieve the PlanarPos structure
131 //
132 // @param   undefined   void
133 // @return  struct planarPos_tag*
134 struct RNA::planarPos_tag* RNA::GetPlanarPos() {
```

```
135     return PlanarPos;
136 }
```

```
1  /*****************************************************
2  * Copyright (c): 2006, All Rights Reserved
3  * Project:       SJSU Masters Project
4  * File:          Nussinov.h
5  * Purpose:       Header file for RNA progress bar
6  *
7  * Start Date:    10/4/2006
8  * Programmer:    Brandon Hunter
9  *****************************************************/
10
11 #pragma once
12
13 class ProgressGraph
14 {
15 private:
16
17 public:
18     ProgressGraph(void);
19     ~ProgressGraph(void);
20
21     void Draw(HDC hdc, char* Sequence, int RNALength, int StepPosition, float xsize,
       float ysize);
22 };
```

```cpp
/*****************************************************
* Copyright (c): 2006, All Rights Reserved
* Project:       SJSU Masters Project
* File:          Nussinov.cpp
* Purpose:       Class implementation of the RNA progress bar
*
* Start Date:    10/4/2006
* Programmer:    Brandon Hunter
*
*****************************************************/

#include "StdAfx.h"
#include "ProgressGraph.h"
#include "math.h" // for the floor function

// FUNCTION:  ProgressGraph
// Default Constructor
//
// @param   undefined   void
// @return  void
ProgressGraph::ProgressGraph(void)
{
}

// FUNCTION:  ~ProgressGraph
// Default Destructor
//
// @param   undefined   void
// @return  void
ProgressGraph::~ProgressGraph(void)
{
}

// FUNCTION:  Draw
// Used to draw the Progress Graph bitmap onto the device context
//
// @param   hdc            HDC    the bitmap device context
// @param   Sequence       char*  the sequence of nucleotide bases
// @param   RNALength      int    the length of the RNA sequence
// @param   StepPosition   int    the current step position
// @param   xsize          float  the x coordinate of the bitmap size
// @param   ysize          float  the y coordinate of the bitmap size
// @return  void
void ProgressGraph::Draw(HDC hdc, char* Sequence, int RNALength, int StepPosition,
    float xsize, float ysize) {
    float fBorder = 5.0f;
    float fCellSizeX = 20;
    float fCellSizeY = 20;
    float fCellLeft = fBorder;
    float fCellTop = fBorder;
    float xPos = fCellLeft, yPos = fCellTop;
    int iCellsPerLine = (int)floor((xsize - (fBorder * 2)) / fCellSizeX); // Number of
     cells that fit on a line
    int iLineCount = RNALength / iCellsPerLine; // Number of complete lines
    int iRemainder = RNALength % iCellsPerLine; // Number of cells remaining on last
    line

    HPEN hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 255));
    HPEN hOldPen = (HPEN)SelectObject(hdc, hPen);
    HBRUSH hBrush = CreateSolidBrush(RGB(128, 128, 128));
    HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);
    Rectangle(hdc, 0, 0, (int)(xsize + 0.5f), (int)(ysize + 0.5f)); // Set the
    background color for the whole facet
    DeleteObject(hBrush);
    hBrush = CreateSolidBrush(RGB(255, 255, 255));
    SelectObject(hdc, hBrush);
    //for(int x = 0; x < iLineCount; x++)
```

```
64      //{
65      //  Rectangle(hdc, (int)xPos, (int)yPos, (int)(xPos + (fCellSizeX * iCellsPerLine)
         + 0.5f), (int)(yPos + fCellSizeY + 0.5f)); // Draw a box the covers the whole
        line
66      //  yPos += fCellSizeY;
67      //}
68      //Rectangle(hdc, (int)xPos, (int)yPos, (int)(xPos + (fCellSizeX * iRemainder) + 0.
        5f), (int)(yPos + fCellSizeY + 0.5f)); // Draw the remaining boxes
69      SelectObject(hdc, hOldBrush);
70      DeleteObject(hBrush);
71      SelectObject(hdc, hOldPen);
72      DeleteObject(hPen);
73
74      // Draw the sequence
75      COLORREF Bkgrnd;
76      COLORREF OldBkgrnd;
77      COLORREF Foregrnd;
78      COLORREF OldForegrnd;
79
80      HFONT hFont = CreateFont((int)fCellSizeY - 2, 0, 0, 0, FW_SEMIBOLD, FALSE, FALSE,
        FALSE, ANSI_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
        FF_DONTCARE, TEXT("Arial"));
81      HFONT hOldFont = (HFONT)SelectObject(hdc, hFont);
82      xPos = fCellLeft;
83      yPos = fCellTop;
84      for(int x = 0; x < RNALength; x++)
85      {
86          if(x > 0 && (x % iCellsPerLine) == 0)
87          {
88              yPos += fCellSizeY;
89              xPos = fCellLeft;
90          }
91
92          char strBase[2];
93          strBase[0] = Sequence[x];
94          strBase[1] = '\0';
95          int len = strlen(strBase);
96
97          if(x < StepPosition)
98          {
99              Foregrnd = RGB(255, 255, 255);
100             Bkgrnd = RGB(49, 106, 197);  //Bkgrnd = RGB(186, 209, 252);
101             RECT rect = {xPos, yPos, xPos + fCellSizeX, yPos + fCellSizeY};
102             HBRUSH hBrush = CreateSolidBrush(Bkgrnd);
103             HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);
104             FillRect(hdc, &rect, hBrush);
105             SelectObject(hdc, hOldBrush);
106             DeleteObject(hBrush);
107         }
108         else
109         {
110             Foregrnd = RGB(0, 0, 0);
111             Bkgrnd = RGB(255, 255, 255);
112             RECT rect = {xPos, yPos, xPos + fCellSizeX, yPos + fCellSizeY};
113             HBRUSH hBrush = CreateSolidBrush(Bkgrnd);
114             HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);
115             FillRect(hdc, &rect, hBrush);
116             SelectObject(hdc, hOldBrush);
117             DeleteObject(hBrush);
118         }
119
120         SIZE sizeRect;
121         GetTextExtentPointA(hdc, strBase, len, &sizeRect);
122         OldForegrnd = SetTextColor(hdc, Foregrnd);
123         OldBkgrnd = SetBkColor(hdc, Bkgrnd);
124         TextOutA(hdc, (int)((xPos + (fCellSizeX / 2.0f)) - (sizeRect.cx / 2.0f) + 0.
    5f), (int)((yPos + (fCellSizeY / 2.0f)) - (sizeRect.cy / 2.0f) + 0.5f), strBase,
```

```
    len);
125         SetTextColor(hdc, OldForegrnd);
126         SetBkColor(hdc, OldBkgrnd);
127 
128         xPos += fCellSizeX;
129     }
130     SelectObject(hdc, hOldFont);
131     DeleteObject(hFont);
132 }
```

```
/***************************************************
* Copyright (c): 2006, All Rights Reserved
* Project:       SJSU Masters Project
* File:          Nussinov.h
* Purpose:       Header file for Nussinov algorithm implementation
*                as described in the book
*                Biological Sequence Analysis, R. Durbin,
*                S. Eddy, A. Krogh, G. Mitchison p. 270-273
*
* Start Date:    8/1/2006
* Programmer:    Brandon Hunter
*
***************************************************/

#include "Stack.h" // Stack used during traceback
#include "List.h" // List to hold the traceback path
#include "RNA.h"

#pragma once

class Nussinov
{
public:
    enum NussinovType
    {
        NussinovStandard = 0,
        NussinovSCFG = 1,
    };

private:
    NussinovType mType; // Standard or SCFG
    char *RNASequence;
    int RNALength;
    int scoreMatrix[4][4]; // Scoring Matrix
    float probMatrix[4][3]; // Probability Matrix
    float probSS; // Bifurcation Probability
    int minLoopLength; // Minimum Hairpin Loop Length
    int** Matrix;
    float** CYKMatrix;
    float Infinity; // use maximum negative float for negative infinity (float = 3.4x10^-38 to 3.4x10^38)

    template<class T> T** createMatrix(int row, int col);
    template<class T> void deleteMatrix(T** mMatrix, int row);
    int Max(int a, int b, int c, int d);
    float Max(float a, float b, float c);
    void NussinovFillStage();
    void NussinovTraceBack(RNA *pRNA);
    void NussinovFillStageCYK();
    void NussinovTraceBackCYK(RNA *pRNA);
    void SortTracebackPath();
    bool IsEqual(float num1, float num2);

public:
    Nussinov(NussinovType mType, char *RNASequence, int RNALength);
    ~Nussinov(void);

    void setScoringMatrix(int AA, int AC, int AG, int AU, int CA, int CC, int CG, int CU, int GA, int GC, int GG, int GU, int UA, int UC, int UG, int UU);
    int getScoringMatrix(int row, int col);
    void setMinHairpinLength(int HairpinLength);
    int getMinHairpinLength(void);
    void setProbMatrix(float aS, float cS, float gS, float uS, float Sa, float Sc, float Sg, float Su, float aSu, float cSg, float gSc, float uSa, float SS);
    float getProbMatrix(int row, int col);
    float getProbSS(void);
    void FillStage();
```

```
65      void TraceBack(RNA *pRNA);
66
67      List listTraceback;  // Instantiate the list which holds traceback path
68
69      int** getMatrix(void);
70      float** getCYKMatrix(void);
71      NussinovType getType(void);
72
73      float getInfinity(void);
74  };
```

```cpp
/*************************************************
* Copyright (c): 2006, All Rights Reserved
* Project:       SJSU Masters Project
* File:          Nussinov.cpp
* Purpose:       Class implementation of both the standard and the
*                SCFG version of the Nussinov folding algorithm
*                as described in the book
*                Biological Sequence Analysis, R. Durbin,
*                S. Eddy, A. Krogh, G. Mitchison p. 270-273
*
* Start Date:    8/1/2006
* Programmer:    Brandon Hunter
*
*************************************************/

#include "StdAfx.h"
#include "Nussinov.h"
#include "math.h"

// FUNCTION:  Nussinov
// Default Constructor
//
// @param   mType         NussinovType   the nussinov type (standard or SCFG)
// @param   RNASequence   char*          the sequence of nucleotide bases
// @param   RNALength     int            the length of the RNA sequence
// @return  void
Nussinov::Nussinov(NussinovType mType, char *RNASequence, int RNALength)
{
    Nussinov::mType = mType; // Set the Nussinov type (Standard or SCFG)

    // Store the RNA Sequence and Length into private memory variables
    Nussinov::RNASequence = RNASequence;
    Nussinov::RNALength = RNALength;

    switch (mType)
    {
    case NussinovStandard:
        Matrix = createMatrix<int>(RNALength, RNALength);
        break;
    case NussinovSCFG:
        CYKMatrix = createMatrix<float>(RNALength, RNALength);
        break;
    default:
        //  It's not a valid value, raise an exception.  We should never get here...
        throw "An invalid Nussinov Type has been encountered.";  // Could use
    RaiseException(ERROR);
        break;
    }

    // Set default scoring matrix
    scoreMatrix[0][0] = 0;
    scoreMatrix[0][1] = 0;
    scoreMatrix[0][2] = 0;
    scoreMatrix[0][3] = 1;
    scoreMatrix[1][0] = 0;
    scoreMatrix[1][1] = 0;
    scoreMatrix[1][2] = 1;
    scoreMatrix[1][3] = 0;
    scoreMatrix[2][0] = 0;
    scoreMatrix[2][1] = 1;
    scoreMatrix[2][2] = 0;
    scoreMatrix[2][3] = 0;
    scoreMatrix[3][0] = 1;
    scoreMatrix[3][1] = 0;
    scoreMatrix[3][2] = 0;
    scoreMatrix[3][3] = 0;

```

```
67      // Set default probability matrix
68      probMatrix[0][0] = log(0.024f); // i - unpaired Probability
69      probMatrix[1][0] = log(0.024f); // i - unpaired Probability
70      probMatrix[2][0] = log(0.024f); // i - unpaired Probability
71      probMatrix[3][0] = log(0.024f); // i - unpaired Probability
72      probMatrix[0][1] = log(0.024f); // j - unpaired Probability
73      probMatrix[1][1] = log(0.024f); // j - unpaired Probability
74      probMatrix[2][1] = log(0.024f); // j - unpaired Probability
75      probMatrix[3][1] = log(0.024f); // j - unpaired Probability
76      probMatrix[0][2] = log(0.2f); // i,j - paired Probability
77      probMatrix[1][2] = log(0.2f); // i,j - paired Probability
78      probMatrix[2][2] = log(0.2f); // i,j - paired Probability
79      probMatrix[3][2] = log(0.2f); // i,j - paired Probability
80      probSS = log(0.008f); // bifurcation probability
81
82      // Set default hairpin loop length
83      minLoopLength = 3;
84
85      Infinity = -274877906944.0f; // use maximum negative float for negative infinity
        (float = 3.4x10^-38 to 3.4x10^38)
86  }
87
88  // FUNCTION:  ~Nussinov
89  // Default Destructor
90  //
91  // @param   undefined   void
92  // @return  void
93  Nussinov::~Nussinov(void) {
94      switch (mType)
95      {
96      case NussinovStandard:
97          deleteMatrix<int>(Matrix, RNALength);
98          break;
99      case NussinovSCFG:
100         deleteMatrix<float>(CYKMatrix, RNALength);
101         break;
102     default:
103         //  It's not a valid value, raise an exception.  We should never get here...
104         throw "An invalid Nussinov Type has been encountered.";  // Could use
        RaiseException(ERROR);
105         break;
106     }
107 }
108
109 // FUNCTION:  getInfinity
110 // Function used to get a number used as infinity
111 //
112 // @param   undefined   void
113 // @return  float
114 float Nussinov::getInfinity(void) {
115     return Infinity;
116 }
117
118 // FUNCTION:  FillStage
119 // Function used to call the desired FillStage (Standard or SCFG)
120 //
121 // @param   undefined   void
122 // @return  void
123 void Nussinov::FillStage() {
124     switch (mType)
125     {
126     case NussinovStandard:
127         NussinovFillStage();
128         break;
129     case NussinovSCFG:
130         NussinovFillStageCYK();
131         break;
```

```
132     default:
133         //  It's not a valid value, raise an exception.  We should never get here...
134         throw "An invalid Nussinov Type has been encountered.";  // Could use
    RaiseException(ERROR);
135         break;
136     }
137 }
138 
139 // FUNCTION:  TraceBack
140 // Function used to call the correct TraceBack (Standard or SCFG)
141 //
142 // @param   pRNA  RNA*  pointer to an RNA class
143 // @return  void
144 void Nussinov::TraceBack(RNA *pRNA) {
145     switch (mType)
146     {
147     case NussinovStandard:
148         NussinovTraceBack(pRNA);
149         break;
150     case NussinovSCFG:
151         NussinovTraceBackCYK(pRNA);
152         break;
153     default:
154         //  It's not a valid value, raise an exception.  We should never get here...
155         throw "An invalid Nussinov Type has been encountered.";  // Could use
    RaiseException(ERROR);
156         break;
157     }
158 }
159 
160 // FUNCTION:  getType
161 // Function used to retrieve the Nussinov type currently in use
162 //
163 // @param   undefined   void
164 // @return  NussinovType
165 Nussinov::NussinovType Nussinov::getType(void) {
166     return mType;
167 }
168 
169 // FUNCTION:  createMatrix
170 // Used to dynamically allocate memory for the Nussinov matrix
171 //
172 // @param   row   int   the number of rows in the matrix
173 // @param   col   int   the number of columns in the matrix
174 // @return  T**
175 template<class T> T** Nussinov::createMatrix(int row, int col) {
176     T** mMatrix = new T* [row];
177 
178     for(int i = 0; i < row; i++)
179         mMatrix[i] = new T[col];
180     return mMatrix;
181 }
182 
183 // FUNCTION:  deleteMatrix
184 // Used to remove the memory that was dynamically allocated for the Nussinov matrix
185 //
186 // @param   mMatrix   T**   pointer to matrix
187 // @param   row       int   number of rows in the matrix
188 // @return  void
189 template<class T> void Nussinov::deleteMatrix(T** mMatrix, int row) {
190     for(int i = 0; i < row; i++) {
191         delete[] mMatrix[i];
192     }
193     delete[] mMatrix;
194 }
195 
196 // FUNCTION:  setScoringMatrix
```

```
197 // Used to populate the scoring matrix
198 //
199 // @param   AA   int   the A-A bond value
200 // @param   AC   int   the A-C bond value
201 // @param   AG   int   the A-G bond value
202 // @param   AU   int   the A-U bond value
203 // @param   CA   int   the C-A bond value
204 // @param   CC   int   the C-C bond value
205 // @param   CG   int   the C-G bond value
206 // @param   CU   int   the C-U bond value
207 // @param   GA   int   the G-A bond value
208 // @param   GC   int   the G-C bond value
209 // @param   GG   int   the G-G bond value
210 // @param   GU   int   the G-U bond value
211 // @param   UA   int   the U-A bond value
212 // @param   UC   int   the U-C bond value
213 // @param   UG   int   the U-G bond value
214 // @param   UU   int   the U-U bond value
215 // @return  void
216 void Nussinov::setScoringMatrix(int AA, int AC, int AG, int AU, int CA, int CC, int CG↙
        , int CU, int GA, int GC, int GG, int GU, int UA, int UC, int UG, int UU) {
217     scoreMatrix[0][0] = AA;
218     scoreMatrix[0][1] = AC;
219     scoreMatrix[0][2] = AG;
220     scoreMatrix[0][3] = AU;
221     scoreMatrix[1][0] = CA;
222     scoreMatrix[1][1] = CC;
223     scoreMatrix[1][2] = CG;
224     scoreMatrix[1][3] = CU;
225     scoreMatrix[2][0] = GA;
226     scoreMatrix[2][1] = GC;
227     scoreMatrix[2][2] = GG;
228     scoreMatrix[2][3] = GU;
229     scoreMatrix[3][0] = UA;
230     scoreMatrix[3][1] = UC;
231     scoreMatrix[3][2] = UG;
232     scoreMatrix[3][3] = UU;
233 }
234
235 // FUNCTION:  getScoringMatrix
236 // Used to retrieve the cell value of the given position
237 //
238 // @param   row   int  the row of the desired cell
239 // @param   col   int  the col of the desired cell
240 // @return  ing
241 int Nussinov::getScoringMatrix(int row, int col) {
242     int Value;
243     if(row >= 0 && row < 4 && col >= 0 && col < 4)
244     {
245         Value = scoreMatrix[row][col];
246     }
247     else
248         throw "Nussinov: Invalid scoring matrix index";
249
250     return Value;
251 }
252
253 // FUNCTION:  setMinHairpinLength
254 // Used to set the minimum hairpin loop length
255 //
256 // @param   HairpinLength   int   the minimum length of the hairpin loop
257 // @return  void
258 void Nussinov::setMinHairpinLength(int HairpinLength) {
259     minLoopLength = HairpinLength;
260 }
261
262 // FUNCTION:  getMinHairpinLength
```

```
263 // Used to retrieve the minimum hairpin loop length
264 //
265 // @param   undefined   void
266 // @return  int
267 int Nussinov::getMinHairpinLength(void) {
268     return minLoopLength;
269 }
270
271 // FUNCTION:  NussinovFillStage
272 // Implementation of the Nussinov FillStage
273 //
274 // @param   undefined   void
275 // @return  void
276 void Nussinov::NussinovFillStage() {
277     for(int i = 0; i < RNALength; i++) { // Initialize the diagonal to zero
278         Matrix[i][i] = 0;
279     }
280     for(int i = 1; i < RNALength; i++) { // Initialize the lower diagonal to zero
281         Matrix[i][i - 1] = 0;
282     }
283
284     int j;
285     for(int d = 1; d < RNALength; d++) { // Loop through the diagonals
286         for(int i = 0; i < RNALength - d; i++) { // Loop within the diagonal
287             j = i + d;
288             if(d <= minLoopLength) {
289                 Matrix[i][j] = 0;
290             }
291             else {
292                 int maxValue = 0;
293                 for(int k = i + 1; k < j; k++) {
294                     int tmpValue = Matrix[i][k] + Matrix[k + 1][j];
295                     if(tmpValue > maxValue)
296                         maxValue = tmpValue;
297                 }
298                 Matrix[i][j] = Max(Matrix[i + 1][j], Matrix[i][j - 1], Matrix[i + 1][j
       - 1] + scoreMatrix[(RNASequence[i] == 'A' ? 0 : (RNASequence[i] == 'C' ? 1 :
     (RNASequence[i] == 'G' ? 2 : 3)))][(RNASequence[j] == 'A' ? 0 : (RNASequence[j] ==
      'C' ? 1 : (RNASequence[j] == 'G' ? 2 : 3)))], maxValue);
299             }
300         }
301     }
302 }
303
304 // FUNCTION:  NussinovTraceBack
305 // Implementation of the Nussinov TraceBack Stage
306 //
307 // @param   pRNA   RNA*   pointer to an RNA class
308 // @return  void
309 void Nussinov::NussinovTraceBack(RNA *pRNA) {
310     CStack stackPosition;  // Instantiate the position stack
311
312     int* result = new int[pRNA->getStepPosition()];
313
314     listTraceback.Clear(); // Clear the old list
315
316     CStack::Item_type item;  // Instantiate a item structure
317     CStack::Item_type* item_ptr = &item; // Pointer to Item_type
318
319     item.i = 0;
320     item.j = pRNA->getStepPosition() - 1;
321
322     stackPosition.Push(item);  // Push the upper right corner of the matrix on the
      stack
323     listTraceback.Add(item.i, item.j); // Traceback stack position
324
325     for(int x = 0; x < pRNA->getStepPosition(); x++) {
```

```
326         result[x] = x;
327     }
328 
329     int i, j;
330     while(!stackPosition.IsEmpty()) {
331         stackPosition.Pop(item_ptr);
332         i = item_ptr->i;
333         j = item_ptr->j;
334 
335         if(i >= j) {
336             continue;   // Continue to next iteration
337         }
338         else if(Matrix[i][j] == Matrix[i + 1][j]) {
339             // i Unpaired
340             item.i = i + 1;
341             item.j = j;
342             stackPosition.Push(item);
343             listTraceback.Add(item.i, item.j);
344         }
345         else if(Matrix[i][j] == Matrix[i][j - 1]) {
346             // j Unpaired
347             item.i = i;
348             item.j = j - 1;
349             stackPosition.Push(item);
350             listTraceback.Add(item.i, item.j);
351         }
352         else if(Matrix[i][j] == Matrix[i + 1][j - 1] + scoreMatrix[(RNASequence[i] == ↙
    'A' ? 0 : (RNASequence[i] == 'C' ? 1 : (RNASequence[i] == 'G' ? 2 : 3)))][      ↙
    (RNASequence[j] == 'A' ? 0 : (RNASequence[j] == 'C' ? 1 : (RNASequence[j] == 'G' ?↙
     2 : 3)))]) {
353             // i, j Paired
354             item.i = i + 1;
355             item.j = j - 1;
356             stackPosition.Push(item);
357             listTraceback.Add(item.i, item.j);
358             result[i] = j;
359             result[j] = i;
360         }
361         else {
362             for(int k = i + 1; k < j; k++) {
363                 if(Matrix[i][j] == Matrix[i][k] + Matrix[k + 1][j]) {
364                     // Bifurcation
365                     item.i = k + 1;
366                     item.j = j;
367                     stackPosition.Push(item);
368                     listTraceback.Add(item.i, item.j);
369                     item.i = i;
370                     item.j = k;
371                     stackPosition.Push(item);
372                     listTraceback.Add(item.i, item.j);
373                     break;
374                 }
375             }
376         }
377     }
378 
379     SortTracebackPath();
380     pRNA->setPairing(result);
381 }
382 
383 // FUNCTION:  SortTracebackPath
384 // Used to sort the traceback path
385 //
386 // @param   undefined   void
387 // @return  void
388 void Nussinov::SortTracebackPath() {
389     List::Node_type *node_ptr = listTraceback.head;
```

```
390     List::Node_type *node_previous = NULL;
391
392     while(node_ptr->next != NULL)
393     {
394         if(node_ptr->i > node_ptr->next->i)
395         {
396             if(node_previous == NULL)
397             {
398                 listTraceback.head = node_ptr->next;
399                 node_previous = node_ptr->next;
400                 node_ptr->next = node_ptr->next->next;
401                 node_previous->next = node_ptr;
402             }
403             else
404             {
405                 node_previous->next = node_ptr->next;
406                 node_ptr->next = node_ptr->next->next;
407                 node_previous->next->next = node_ptr;
408                 node_previous = node_previous->next;
409             }
410         }
411         else
412         {
413             node_previous = node_ptr;
414             node_ptr = node_ptr->next;
415         }
416     }
417     listTraceback.tail = node_ptr;
418 }
419
420 // FUNCTION:  getMatrix
421 // Used to retrieve the dynamic programming matrix
422 //
423 // @param   undefined   void
424 // @return  int**
425 int** Nussinov::getMatrix(void) {
426     return Matrix;
427 }
428
429 // FUNCTION:  getCYKMatrix
430 // Used to retrieve the CYK dynamic programming matrix
431 //
432 // @param   undefined   void
433 // @return  float**
434 float** Nussinov::getCYKMatrix(void) {
435     return CYKMatrix;
436 }
437
438 // FUNCTION:  Max
439 // Returns the maximum of the 4 integers
440 //
441 // @param   a   int   first integer
442 // @param   b   int   second integer
443 // @param   c   int   third integer
444 // @param   d   int   fourth integer
445 // @return  int
446 int Nussinov::Max(int a, int b, int c, int d) {
447     int tmp = a;
448     if(b > tmp)
449         tmp = b;
450     if(c > tmp)
451         tmp = c;
452     if(d > tmp)
453         tmp = d;
454
455     return tmp;
456 }
```

```
457
458 // FUNCTION:  Max
459 // Returns the maximum of 3 floats
460 //
461 // @param   a   float   the first float
462 // @param   b   float   the second float
463 // @param   c   float   the third float
464 // @return  float
465 float Nussinov::Max(float a, float b, float c) {
466     float tmp = a;
467     if(b > tmp)
468         tmp = b;
469     if(c > tmp)
470         tmp = c;
471
472     return tmp;
473 }
474
475 // FUNCTION:  setProbMatrix
476 // Used to set the probability matrix
477 //
478 // @param   aS   float   the aS probability
479 // @param   cS   float   the cS probability
480 // @param   gS   float   the gS probability
481 // @param   uS   float   the uS probability
482 // @param   Sa   float   the Sa probability
483 // @param   Sc   float   the Sc probability
484 // @param   Sg   float   the Sg probability
485 // @param   Su   float   the Su probability
486 // @param   aSu   float   the aSu probability
487 // @param   cSg   float   the cSg probability
488 // @param   gSc   float   the gSc probability
489 // @param   uSa   float   the uSa probability
490 // @param   SS    float   the SS probability
491 // @return  void
492 void Nussinov::setProbMatrix(float aS, float cS, float gS, float uS, float Sa, float ↙
        Sc, float Sg, float Su, float aSu, float cSg, float gSc, float uSa, float SS) {
493     probMatrix[0][0] = log(aS);
494     probMatrix[1][0] = log(cS);
495     probMatrix[2][0] = log(gS);
496     probMatrix[3][0] = log(uS);
497     probMatrix[0][1] = log(Sa);
498     probMatrix[1][1] = log(Sc);
499     probMatrix[2][1] = log(Sg);
500     probMatrix[3][1] = log(Su);
501     probMatrix[0][2] = log(aSu);
502     probMatrix[1][2] = log(cSg);
503     probMatrix[2][2] = log(gSc);
504     probMatrix[3][2] = log(uSa);
505     probSS = log(SS);
506 }
507
508 // FUNCTION:  getProbMatrix
509 // Used to retrieve the probability matrix
510 //
511 // @param   row   int   the row of the cell to retreive
512 // @param   col   int   the col of the cell to retreive
513 // @return  float
514 float Nussinov::getProbMatrix(int row, int col) {
515     float Value;
516     if(row >= 0 && row < 4 && col >= 0 && col < 3)
517     {
518         Value = probMatrix[row][col];
519     }
520     else
521         throw "Nussinov: Invalid probability matrix index";
522
```

```
523     return Value;
524 }
525
526 // FUNCTION:  getProbSS
527 // Used to retrieve the SS probability
528 //
529 // @param   undefined   void
530 // @return  float
531 float Nussinov::getProbSS(void) {
532     return probSS;
533 }
534
535 // FUNCTION:  NussinovFillStageCYK
536 // Implementation of the Nussinov CYK FillStage
537 //
538 // @param   undefined   void
539 // @return  void
540 void Nussinov::NussinovFillStageCYK() {
541     for(int i = 0; i < RNALength; i++) { // Initialize the diagonal
542         if(probMatrix[(RNASequence[i] == 'A' ? 0 : (RNASequence[i] == 'C' ? 1 :
      (RNASequence[i] == 'G' ? 2 : 3)))][0] > probMatrix[(RNASequence[i] == 'A' ? 0 :
      (RNASequence[i] == 'C' ? 1 : (RNASequence[i] == 'G' ? 2 : 3)))][1])
543             CYKMatrix[i][i] = (float)probMatrix[(RNASequence[i] == 'A' ? 0 :
      (RNASequence[i] == 'C' ? 1 : (RNASequence[i] == 'G' ? 2 : 3)))][0];
544         else
545             CYKMatrix[i][i] = (float)probMatrix[(RNASequence[i] == 'A' ? 0 :
      (RNASequence[i] == 'C' ? 1 : (RNASequence[i] == 'G' ? 2 : 3)))][1];
546         //CYKMatrix[i][i] = max(probMatrix[(RNASequence[i] == 'A' ? 0 : (RNASequence
      [i] == 'C' ? 1 : (RNASequence[i] == 'G' ? 2 : 3)))][0], probMatrix[(RNASequence[i]
       == 'A' ? 0 : (RNASequence[i] == 'C' ? 1 : (RNASequence[i] == 'G' ? 2 : 3)))][1]);
547     }
548     for(int i = 1; i < RNALength; i++) { // Initialize the lower diagonal to negative
      infinity
549         CYKMatrix[i][i - 1] = -274877906944; // use maximum negative float for
      negative infinity (float = 3.4x10^-38 to 3.4x10^38)
550     }
551
552     int j;
553     for(int d = 1; d < RNALength; d++) { // Loop through the diagonals
554         for(int i = 0; i < RNALength - d; i++) { // Loop within the diagonal
555             j = i + d;
556             if(d <= minLoopLength) {
557                 CYKMatrix[i][j] = 0;
558             }
559             else {
560                 float maxValue = Infinity; // use maximum negative float for negative
      infinity (float = 3.4x10^-38 to 3.4x10^38)
561                 for(int k = i + 1; k < j; k++) {
562                     float tmpValue = CYKMatrix[i][k] + CYKMatrix[k + 1][j] + probSS;
563                     if(tmpValue > maxValue)
564                         maxValue = tmpValue;
565                 }
566                 if((RNASequence[i] == 'A' ? 0 : (RNASequence[i] == 'C' ? 1 :
      (RNASequence[i] == 'G' ? 2 : 3))) == 3 - (RNASequence[j] == 'A' ? 0 : (RNASequence
      [j] == 'C' ? 1 : (RNASequence[j] == 'G' ? 2 : 3)))) // A-U, U-A, C-G, G-C
567                 {
568                     float tmpValue = CYKMatrix[i+1][j-1] + probMatrix[(RNASequence[i]
      == 'A' ? 0 : (RNASequence[i] == 'C' ? 1 : (RNASequence[i] == 'G' ? 2 : 3)))][2];
569                     if(tmpValue > maxValue)
570                         maxValue = tmpValue;
571                 }
572                 if(i == 0 && j == 8)
573                     i = i;
574                 CYKMatrix[i][j] = Max(CYKMatrix[i + 1][j] + probMatrix[(RNASequence[i]
       == 'A' ? 0 : (RNASequence[i] == 'C' ? 1 : (RNASequence[i] == 'G' ? 2 : 3)))][0],
      CYKMatrix[i][j - 1] + probMatrix[(RNASequence[j] == 'A' ? 0 : (RNASequence[j] ==
      'C' ? 1 : (RNASequence[j] == 'G' ? 2 : 3)))][1], maxValue);
```

```
575             }
576         }
577     }
578 }
579
580 // FUNCTION:  NussinovTraceBackCYK
581 // Implementation of the Nussinov CYK TraceBack Stage
582 //
583 // @param   pRNA   RNA*   pointer to an RNA class
584 // @return  void
585 void Nussinov::NussinovTraceBackCYK(RNA *pRNA) {
586     CStack stack_pos;  // Instantiate the position stack
587
588     int* result = new int[pRNA->getStepPosition()];
589
590     listTraceback.Clear(); // Clear the old list
591
592     CStack::Item_type item;  // Instantiate a item structure
593     CStack::Item_type* item_ptr = &item; // Pointer to Item_type
594
595     item.i = 0;
596     item.j = pRNA->getStepPosition() - 1;
597
598     stack_pos.Push(item);  // Push the item on the stack
599     listTraceback.Add(item.i, item.j); // Traceback stack position
600
601     int i, j;
602     for(i = 0; i < pRNA->getStepPosition(); i++) {
603         result[i] = i;
604     }
605
606     while(!stack_pos.IsEmpty()) {
607         stack_pos.Pop(item_ptr);
608         i = item_ptr->i;
609         j = item_ptr->j;
610
611         if(i >= j) {
612             continue;   // Continue to next iteration
613         }
614         else if(IsEqual(CYKMatrix[i][j], (float)(CYKMatrix[i + 1][j] + probMatrix[
    (RNASequence[i] == 'A' ? 0 : (RNASequence[i] == 'C' ? 1 : (RNASequence[i] == 'G' ?
     2 : 3)))][0]))) {
615             item.i = i + 1;
616             item.j = j;
617             stack_pos.Push(item);
618             listTraceback.Add(item.i, item.j);
619         }
620         else if(IsEqual(CYKMatrix[i][j], (float)(CYKMatrix[i][j - 1] + probMatrix[
    (RNASequence[j] == 'A' ? 0 : (RNASequence[j] == 'C' ? 1 : (RNASequence[j] == 'G' ?
     2 : 3)))][1]))) {
621             item.i = i;
622             item.j = j - 1;
623             stack_pos.Push(item);
624             listTraceback.Add(item.i, item.j);
625         }
626         else if(((RNASequence[i] == 'A' ? 0 : (RNASequence[i] == 'C' ? 1 :
    (RNASequence[i] == 'G' ? 2 : 3))) == 3 - (RNASequence[j] == 'A' ? 0 : (RNASequence
    [j] == 'C' ? 1 : (RNASequence[j] == 'G' ? 2 : 3)))) && (IsEqual(CYKMatrix[i][j],
    (float)(CYKMatrix[i + 1][j - 1] + probMatrix[(RNASequence[i] == 'A' ? 0 :
    (RNASequence[i] == 'C' ? 1 : (RNASequence[i] == 'G' ? 2 : 3)))][2])))) {
627             item.i = i + 1;
628             item.j = j - 1;
629             stack_pos.Push(item);
630             listTraceback.Add(item.i, item.j);
631             result[i] = j;
632             result[j] = i;
633         }
```

```
634         else {
635             for(int k = i + 1; k < j; k++) {
636                 if(IsEqual(CYKMatrix[i][j], (float)(CYKMatrix[i][k] + CYKMatrix[k + 1]↙
    [j] + probSS))) {
637                     item.i = k + 1;
638                     item.j = j;
639                     stack_pos.Push(item);
640                     listTraceback.Add(item.i, item.j);
641                     item.i = i;
642                     item.j = k;
643                     stack_pos.Push(item);
644                     listTraceback.Add(item.i, item.j);
645                     break;
646                 }
647             }
648         }
649     }
650 
651     SortTracebackPath();
652     pRNA->setPairing(result);
653 }
654 
655 // FUNCTION:  IsEqual
656 // Determine if the numbers are within a thousandth of each other, If
657 // they are then consider them equal
658 //
659 // @param   num1   float   the first number to compare
660 // @param   num2   float   the second number to compare
661 // @return  bool
662 bool Nussinov::IsEqual(float num1, float num2)
663 {
664     float diff;
665     if(num1 > num2)
666     {
667         diff = num1 - num2;
668     }
669     else
670     {
671         diff = num2 - num1;
672     }
673 
674     if(diff < .0001)
675         return true;
676     else
677         return false;
678 }
```

```
1  /*****************************************************
2  * Copyright (c): 2006, All Rights Reserved
3  * Project:       SJSU Masters Project
4  * File:          MatrixGraph.cpp
5  * Purpose:       Header File to matrix graph class
6  *
7  * Start Date:    9/15/2006
8  * Programmer:    Brandon Hunter
9  *
10 *****************************************************/
11
12 #pragma once
13
14 #include "RNA.h"
15 #include "Nussinov.h"
16
17 class MatrixGraph
18 {
19 private:
20
21 public:
22     MatrixGraph(void);
23     ~MatrixGraph(void);
24
25     void Draw(HDC hdc, RNA *pRNA, Nussinov *pNussinov, float xsize, float ysize);
26 };
```

```
/*****************************************************
* Copyright (c): 2006, All Rights Reserved
* Project:       SJSU Masters Project
* File:          MatrixGraph.cpp
* Purpose:       To calculate the matrix graph representation of Nussinov Matrix
*
* Start Date:    9/15/2006
* Programmer:    Brandon Hunter
*
*****************************************************/

#include "StdAfx.h"
#include "MatrixGraph.h"
#include "Stack.h"
#include "stdio.h" // for the sprintf function used to convert float to char*

// FUNCTION:  MatrixGraph
// Default Constructor
//
// @param   undefined   void
// @return  void
MatrixGraph::MatrixGraph(void) {
}

// FUNCTION:  ~MatrixGraph
// Default Destructor
//
// @param   undefined   void
// @return  void
MatrixGraph::~MatrixGraph(void) {
}

// Rounding Error Solution used in this function
// int i;
// float f = 1.2345678;
// i = (int)(f + 0.5); /* intValue will be 1 */
// f = 1.56789;
// i = (int)(f + 0.5); /* intValue will be 2 */
// Ex. (int)1.5789 will return 1
//     (int)(1.5789 + .5) will return 2

// FUNCTION:  Draw
// Function used to draw the Matrix graph onto the bitmap device context
//
// @param   hdc         HDC         handle to the bitmap device context
// @param   pRNA        RNA*        pointer to an RNA class
// @param   pNussinov   Nussinov*   pointer to an Nussinov class
// @param   xsize       float       x coordinate of bitmap size
// @param   ysize       float       y coordinate of bitmap size
// @return  void
void MatrixGraph::Draw(HDC hdc, RNA *pRNA, Nussinov *pNussinov, float xsize, float
    ysize) {
    float fBorder = 5.0f;
    float fCellSizeX = (xsize - (2.0f * fBorder)) / (pRNA->getLength() + 1);
    float fCellSizeY = (ysize - (2.0f * fBorder)) / (pRNA->getLength() + 1);
    float fCellLeft = fBorder;
    float fCellTop = fBorder;
    int FontHeight = 0, FontWidth = 0;

    COLORREF Bkgrnd;
    COLORREF OldBkgrnd;
    HFONT hFont;
    HFONT hOldFont;
    HBRUSH hBrush;
    HBRUSH hOldBrush;

    float colorInc = 256.0f / pRNA->getLength(); // pNussinov->listTraceback.count;
```

```
67
68      // Draw the sequence along the edges of the matrix
69      FontHeight = (int)(fCellSizeY - 2.0f + 0.5f);
70      if(FontHeight > 5)
71      {
72          hFont = CreateFont(FontHeight, 0, 0, 0, FW_SEMIBOLD, FALSE, FALSE, FALSE,
        ANSI_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
        FF_DONTCARE, TEXT("Arial"));
73          hOldFont = (HFONT)SelectObject(hdc, hFont);
74          float xPos = fCellLeft + fCellSizeX, yPos = fCellTop + fCellSizeY;
75          for(int x = 0; x < pRNA->getLength(); x++)
76          {
77              char strBase[2];
78              strBase[0] = pRNA->getSequence()[x];
79              strBase[1] = '\0';
80              int len = strlen(strBase);
81
82              SIZE sizeRect;
83              GetTextExtentPointA(hdc, strBase, len, &sizeRect);
84              // Sequence Along the Top Edge
85              TextOutA(hdc, (int)((xPos + (fCellSizeX / 2.0f)) - (sizeRect.cx / 2.0f) +
        0.5f), (int)((fCellTop + (fCellSizeY / 2.0f)) - (sizeRect.cy / 2.0f) + 0.5f),
        strBase, len);
86              // Sequence Along the Left Edge
87              TextOutA(hdc, (int)((fCellLeft + (fCellSizeX / 2.0f)) - (sizeRect.cx / 2.
        0f) + 0.5f), (int)((yPos + (fCellSizeY / 2.0f)) - (sizeRect.cy / 2.0f) + 0.5f),
        strBase, len);
88
89              xPos += fCellSizeX;
90              yPos += fCellSizeY;
91          }
92          SelectObject(hdc, hOldFont);
93          DeleteObject(hFont);
94      }
95
96      fCellLeft += fCellSizeX;
97      fCellTop += fCellSizeY;
98
99      for(int j = 0; j < pRNA->getLength(); j++)
100     {
101         for(int i = 0; i < pRNA->getLength(); i++)
102         {
103             if(j > i - 2) {
104                 if(j >= pRNA->getStepPosition() || i >= pRNA->getStepPosition())
105                 {
106                     Bkgrnd = RGB(224, 224, 224); // Gray background
107
108                 }
109                 else
110                 {
111                     //Bkgrnd = RGB(250, 225, 178);
112                     //Bkgrnd = RGB(colorInc * i, 255, colorInc * j); // Color based on
         color scale
113                     Bkgrnd = RGB(255, 255, 255);
114                     List::Node_type *node_ptr = pNussinov->listTraceback.head;
115                     while(node_ptr != NULL)
116                     {
117                         if(node_ptr->i == i && node_ptr->j == j)
118                         {
119                             Bkgrnd = RGB(colorInc * i, 255, colorInc * j); // Color
        based on color scale
120
121                             // Record postion of trace back cell so lines can be drawn
         from matrix to planar graph
122                             node_ptr->xPos = fCellLeft + (fCellSizeX / 2.0f) + 1.0f;
123                             node_ptr->yPos = fCellTop + (fCellSizeY / 2.0f) + 1.0f;
124
```

```
125                             break;
126                         }
127                         node_ptr = node_ptr->next;
128                     }
129                 }
130                 hBrush = CreateSolidBrush(Bkgrnd); // Gray background
131                 hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);
132                 Rectangle(hdc, (int)(fCellLeft + 0.5f), (int)(fCellTop + 0.5f), (int)
    (fCellLeft + fCellSizeX + 1.0f + 0.5f), (int)(fCellTop + fCellSizeY + 1.0f + 0.
    5f));
133                 SelectObject(hdc, hOldBrush);
134                 DeleteObject(hBrush);
135 
136                 if(j < pRNA->getStepPosition() && i < pRNA->getStepPosition())
137                 {
138                     char strNumber[64]; // Buffer for number
139                     switch(pNussinov->getType())
140                     {
141                     case Nussinov::NussinovType::NussinovStandard:
142                         itoa(pNussinov->getMatrix()[i][j], strNumber, 10);
143                         break;
144                     case Nussinov::NussinovType::NussinovSCFG:
145                         if(pNussinov->getCYKMatrix()[i][j] == pNussinov->getInfinity
    ())
146                         {
147                             strNumber[0] = '-';
148                             strNumber[1] = '\0';
149                         }
150                         else
151                             sprintf(strNumber, "%.2f", pNussinov->getCYKMatrix()[i]
    [j]);
152                         break;
153                     default:
154                         //  It's not a valid value, raise an exception.  We should
    never get here...
155                         throw "An invalid Nussinov Type has been encountered.";  //
    Could use RaiseException(ERROR);
156                         break;
157                     }
158                     int len = strlen(strNumber);
159 
160                     FontHeight = 0;
161                     FontWidth = 0;
162                     if(len > 1) // Need to reduce the size of the font depending on
    the length of the number
163                     {
164                         //hFont = CreateFont((int)fCellSizeY - 5, 0, 0, 0, FW_SEMIBOLD
    , FALSE, FALSE, FALSE, ANSI_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
    DEFAULT_QUALITY, FF_DONTCARE, TEXT("Arial"));
165 
166                         FontWidth = (int)((fCellSizeX / len) - 2.0f + 0.5f);
167                     }
168                     else
169                     {
170                         FontHeight = (int)(fCellSizeY - 2.0f + 0.5f);
171                     }
172                     hFont = CreateFont(FontHeight, FontWidth, 0, 0, FW_SEMIBOLD, FALSE
    , FALSE, FALSE, ANSI_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
    DEFAULT_QUALITY, FF_DONTCARE, TEXT("Arial"));
173                     if(hFont != NULL)
174                         hOldFont = (HFONT)SelectObject(hdc, hFont);
175 
176                     SIZE sizeRect;
177                     GetTextExtentPointA(hdc, strNumber, len, &sizeRect);
178                     OldBkgrnd = SetBkColor(hdc, Bkgrnd);
179                     if(FontWidth > 5 || FontHeight > 5)
180                         TextOutA(hdc, (int)((fCellLeft + (fCellSizeX / 2.0f)) -
```

```
    (sizeRect.cx / 2.0f) + 0.5f), (int)((fCellTop + (fCellSizeY / 2.0f)) - (sizeRect. ↙
    cy / 2.0f) + 0.5f), strNumber, len);
181                     SetBkColor(hdc, OldBkgrnd);
182
183                     SelectObject(hdc, hOldFont);
184                     DeleteObject(hFont);
185                 }
186             }
187
188             fCellTop += fCellSizeY;
189         }
190         fCellTop = fBorder + fCellSizeY;
191         fCellLeft += fCellSizeX;
192     }
193 }
```

```
1  /*****************************************************
2  * Copyright (c): 2006, All Rights Reserved
3  * Project:       SJSU Masters Project
4  * File:          List.h
5  * Purpose:       Header file for class implementation of a linked list
6  *
7  * Start Date:    9/22/2006
8  * Programmer:    Brandon Hunter
9  *
10 *****************************************************/
11
12 #pragma once
13
14 class List
15 {
16 private:
17
18 public:
19     List(void);
20     ~List(void);
21
22     typedef struct node_tag {
23         int i, j;
24         float xPos, yPos;
25         struct node_tag *next;
26     } Node_type;
27
28     void Add(int i, int j);
29     void Clear(void);
30
31     Node_type *head, *tail;
32     int count;
33 };
```

```
/*****************************************************
* Copyright (c): 2006, All Rights Reserved
* Project:       SJSU Masters Project
* File:          List.cpp
* Purpose:       Class implementation of a standard linked list
*
* Start Date:    9/22/2006
* Programmer:    Brandon Hunter
*
*****************************************************/

#include "StdAfx.h"
#include "List.h"

// FUNCTION:  List
// Default Constructor
//
// @param   undefined   void
// @return  void
List::List(void)
{
    head = tail = NULL; // Initialize the head and tail of the list
    count = 0; // Initialize the counter for number of items in list
}

// FUNCTION:  ~List
// Default Destructor
//
// @param   undefined   void
// @return  void
List::~List(void)
{
    while(head != NULL) {
        Node_type *node_ptr;
        node_ptr = head;
        head = node_ptr->next;
        delete node_ptr;
    }
    count = 0;
    head = tail = NULL;
}

// FUNCTION:  Add
// Function to add items to the linked list
//
// @param   i    int    the i pair of the position
// @param   j    int    the j pair of the position
// @return  void
void List::Add(int i, int j) {
    Node_type *p;

    if((p = new Node_type) == NULL)
        throw "List: Out of Memory";
    else {
        p->i = i;
        p->j = j;
        p->next = NULL;
    }

    if(head == NULL)
    {
        head = p; // If the list was empty then set the head and tail to first item
        tail = p;
    }
    else
    {
        tail->next = p;
```

```
68          tail = p;
69      }
70
71      count++; // Increment the list item counter
72  }
73
74  // FUNCTION:  Clear
75  // Function used to clear the linked list
76  //
77  // @param   undefined   void
78  // @return  void
79  void List::Clear(void) {
80      while(head != NULL) {
81          Node_type *node_ptr;
82          node_ptr = head;
83          head = node_ptr->next;
84          delete node_ptr;
85      }
86      count = 0;
87      head = tail = NULL;
88  }
```

```
1  /*****************************************************
2  * Copyright (c): 2006, All Rights Reserved
3  * Project:       SJSU Masters Project
4  * File:          LineList.h
5  * Purpose:       Header file for class implementation of a linked list
6  *
7  * Start Date:    10/22/2006
8  * Programmer:    Brandon Hunter
9  *
10 *****************************************************/
11
12 #pragma once
13
14 class LineList
15 {
16 private:
17
18 public:
19     LineList(void);
20     ~LineList(void);
21
22     typedef struct node_tag {
23         float startX, startY;
24         float endX, endY;
25         DWORD color;
26         struct node_tag *next;
27     } Node_type;
28
29     void Add(float startX, float startY, float endX, float endY, DWORD color);
30     void Clear(void);
31
32     Node_type *head, *tail;
33     int count;
34 };
```

```
/*****************************************************
* Copyright (c): 2006, All Rights Reserved
* Project:       SJSU Masters Project
* File:          LineList.cpp
* Purpose:       Class implementation of a standard linked list
*
* Start Date:    10/22/2006
* Programmer:    Brandon Hunter
*
*****************************************************/

#include "StdAfx.h"
#include "LineList.h"

// FUNCTION:  LineList
// Default Constructor
//
// @param   undefined   void
// @return  void
LineList::LineList(void)
{
    head = tail = NULL; // Initialize the head and tail of the list
    count = 0; // Initialize the counter for number of items in list
}

// FUNCTION:  ~LineList
// Default Destructor
//
// @param   undefined   void
// @return  void
LineList::~LineList(void)
{
    while(head != NULL) {
        Node_type *node_ptr;
        node_ptr = head;
        head = node_ptr->next;
        delete node_ptr;
    }
    count = 0;
    head = tail = NULL;
}

// FUNCTION:  Add
// Function used to add items to the linked list
//
// @param   startX    float    the x coordinate of the start of the line
// @param   startY    float    the y coordinate of the start of the line
// @param   endX      float    the x coordinate of the end of the line
// @param   endY      float    the y coordinate of the end of the line
// @param   color     DWORD    the color of the line
// @return  void
void LineList::Add(float startX, float startY, float endX, float endY, DWORD color) {
    Node_type *p;

    if((p = new Node_type) == NULL)
        throw "List: Out of Memory";
    else {
        p->startX = startX;
        p->startY = startY;
        p->endX = endX;
        p->endY = endY;
        p->color = color;
        p->next = NULL;
    }

    if(head == NULL)
    {
```

```
68          head = p; // If the list was empty then set the head and tail to first item
69          tail = p;
70      }
71      else
72      {
73          tail->next = p;
74          tail = p;
75      }
76
77      count++; // Increment the list item counter
78  }
79
80  // FUNCTION:  Clear
81  // Function that clears the linked list
82  //
83  // @param   undefined   void
84  // @return  void
85  void LineList::Clear(void) {
86      while(head != NULL) {
87          Node_type *node_ptr;
88          node_ptr = head;
89          head = node_ptr->next;
90          delete node_ptr;
91      }
92      count = 0;
93      head = tail = NULL;
94  }
```

```
1  /*****************************************************
2  * Copyright (c): 2006, All Rights Reserved
3  * Project:       SJSU Masters Project
4  * File:          CircularGraph.cpp
5  * Purpose:       Header File to circular graph class
6  *
7  * Start Date:    9/15/2006
8  * Programmer:    Brandon Hunter
9  *
10 *****************************************************/
11
12 #pragma once
13
14 class CircularGraph
15 {
16 private:
17     // Floating point
18     typedef struct tagPOINTF
19     {
20         float x;
21         float y;
22     } POINTF;
23
24 public:
25     CircularGraph(void);
26     ~CircularGraph(void);
27
28     void Draw(HDC hdc, char* Sequence, int iStep, int* Pairing,  float xsize, float
       ysize);
29 };
```

```cpp
/*****************************************************
* Copyright (c): 2006, All Rights Reserved
* Project:       SJSU Masters Project
* File:          CircularGraph.cpp
* Purpose:       To calculate the ciruclar graph representation of RNA secondary structure
*
* Start Date:    9/16/2006
* Programmer:    Brandon Hunter
*
*****************************************************/

#include "StdAfx.h"
#include "CircularGraph.h"
#include "math.h"

// FUNCTION:  CircularGraph
// Default Constructor
//
// @param   undefined   void
// @return  void
CircularGraph::CircularGraph(void) {
}

// FUNCTION:  ~CircularGraph
// Default Destructor
//
// @param   undefined   void
// @return  void
CircularGraph::~CircularGraph(void) {
}

// FUNCTION:  Draw
// This function draws the Circular Graph onto the bitmap device context
//
// @param   hdc         HDC      handle to the bitmap device context
// @param   Sequence    char*    the RNA sequence of nucleotide bases
// @param   iStep       int      the current step position within the sequence
// @param   Pairing     int*     array which holds the sequence pairing positions
// @param   xsize       float    x dimension of bitmap size
// @param   ysize       float    y dimension of bitmap size
// @return  void
void CircularGraph::Draw(HDC hdc, char* Sequence, int iStep, int* Pairing,  float xsize, float ysize) {
    float fBorder = 5.0f;
    float fPadding = 20.0f;
    float fPI = 3.14159265f;
    float fAngle = (360.0f / iStep) * fPI / 180.0f;  // Radians
    float fRadius;
    float fCenterX = 0.0f;
    float fCenterY = 0.0f;

    if(xsize < ysize)
        fRadius = (xsize / 2.0f) - fBorder - fPadding;
    else
        fRadius = (ysize / 2.0f) - fBorder - fPadding;

    HPEN hPen;
    HPEN hOldPen;

    // Draw the tick marks around the circle
    hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 255));
    hOldPen = (HPEN)SelectObject(hdc, hPen);
    for(int x = 0; x < iStep; x++) {
        if((x % 10) == 0) // Draw a slightly longer line on every 10th line
            MoveToEx(hdc, (int)(((fRadius + 10.0f) * cos(x * fAngle)) + fCenterX + 0.5f), (int)(((fRadius + 10.0f) * sin(x * fAngle)) + fCenterY + 0.5f), NULL);
```

```
65          else
66              MoveToEx(hdc, (int)(((fRadius + 5.0f) * cos(x * fAngle)) + fCenterX + 0.5f), (int)(((fRadius + 5.0f) * sin(x * fAngle)) + fCenterY + 0.5f), NULL);
67
68          LineTo(hdc, (int)(fCenterX + 0.5f), (int)(fCenterY + 0.5f));
69      }
70      SelectObject(hdc, hOldPen);
71      DeleteObject(hPen);
72
73      // This is the main cirle
74      Ellipse(hdc, (int)-(fRadius + 0.5f), (int)(fRadius + 0.5f), (int)(fRadius + 0.5f), (int)-(fRadius + 0.5f));
75
76      // Draw tick marks at positions that don't requre arcs
77      hPen = CreatePen(PS_SOLID, 1, RGB(255, 0, 0));
78      hOldPen = (HPEN)SelectObject(hdc, hPen);
79      for(int x = 0; x < iStep; x++) {
80          // Draw tick marks at positions that don't require arcs
81          if(Pairing[x] == x)
82          {
83              MoveToEx(hdc, (int)((fRadius * cos(x * fAngle)) + fCenterX + 0.5f), (int)((fRadius * sin(x * fAngle)) + fCenterY + 0.5f), NULL);
84              LineTo(hdc, (int)(fCenterX + 0.5f), (int)(fCenterY + 0.5f));
85          }
86      }
87      SelectObject(hdc, hOldPen);
88      DeleteObject(hPen);
89
90      // This circle masks the interior of the tick marks
91      hPen = CreatePen(PS_SOLID, 1, RGB(255, 255, 255));
92      hOldPen = (HPEN)SelectObject(hdc, hPen);
93      Ellipse(hdc, (int)(-(fRadius - 5.0f + 0.5f)), (int)(fRadius - 5.0f + 0.5f), (int)(fRadius - 5.0f + 0.5f), (int)(-(fRadius - 5.0f + 0.5f)));
94      SelectObject(hdc, hOldPen);
95      DeleteObject(hPen);
96
97      // Draw the arcs
98      hPen = CreatePen(PS_SOLID, 1, RGB(255, 0, 0));
99      hOldPen = (HPEN)SelectObject(hdc, hPen);
100     for(int x = 0; x < iStep; x++) {
101         // If an arc is required
102         if(Pairing[x] != x) {
103             // Draw the arc when you have reached the second endpoint of the arc
104             if(Pairing[x] < x) {
105                 int i = Pairing[x];
106                 int j = x;
107                 POINTF p1 = {(fRadius * cos(i * fAngle)) + fCenterX, (fRadius * sin(i * fAngle)) + fCenterY};
108                 POINTF p2 = {(fRadius * cos(j * fAngle)) + fCenterX, (fRadius * sin(j * fAngle)) + fCenterY};
109
110                 // Find angle half way between the two tick marks
111                 // Angle = (smaller angle + ((larger angle - smaller angle)) / 2)
112                 float fMidAngle = (i * fAngle) + (((j * fAngle) - (i * fAngle)) / 2.0f);
113
114                 // p3 is 90degrees back from mid angle
115                 POINTF p3 = {(fRadius * cos(fMidAngle - (fPI / 2.0f))) + fCenterX, (fRadius * sin(fMidAngle - (fPI / 2.0f))) + fCenterY};
116                 // p4 is 90degrees forward from mid angle
117                 POINTF p4 = {(fRadius * cos(fMidAngle + (fPI / 2.0f))) + fCenterX, (fRadius * sin(fMidAngle + (fPI / 2.0f))) + fCenterY};
118
119                 if((float)((x - Pairing[x]) * fAngle) == (float)fPI)
120                 {
121                     MoveToEx(hdc, (int)(p1.x + 0.5f), (int)(p1.y + 0.5f), NULL);
122                     LineTo(hdc, (int)(p2.x + 0.5f), (int)(p2.y + 0.5f));
```

```
123                 }
124                 else
125                 {
126                     // Find intersection of lines P2P3 and p1p4 use equation of lines
127                     // and then solve the two equations (two equations two unknowns)
128                     // y = m1x + b1
129 
130                     // m1 = (y2-y1)/(x2-x1)
131                     bool line1Vertical = false;
132                     float m1, b1, m6;
133                     if((p2.x - p3.x) == 0) // Avoid Divide by zero
134                     {
135                         line1Vertical = true; // The line p2p3 is a vertical line
    (slope is undefined)
136                         m6 = 0; // Slope of a line perpendicular to a vertical line is
     0
137                     }
138                     else
139                     {
140                         m1 = (p2.y - p3.y) / (p2.x - p3.x);
141                         b1 = p2.y - (m1 * p2.x); // b1 = y - m1x
142 
143                         m6 = (-1 / m1); // Slope of perpendicular line is m = -1/m1
144                     }
145 
146                     // m2 = (y2-y1)/(x2-x1)
147                     bool line2Vertical = false;
148                     float m2, b2, m5;
149                     if((p1.x - p4.x) == 0) // Avoid Divide by zero
150                     {
151                         line2Vertical = true; // The line p1p4 is a vertical line
    (slope is undefined)
152                         m5 = 0; // Slope of a line perpendicular to a vertical line is
     0
153                     }
154                     else
155                     {
156                         m2 = (p1.y - p4.y) / (p1.x - p4.x);
157                         b2 = p1.y - (m2 * p1.x); // b2 = y - m2x
158 
159                         m5 = (-1 / m2); // Slope of perpendicular line is m = -1/m2
160                     }
161 
162                     // Now find intersection of p1p4 and p2p3
163                     POINTF px;
164                     if(line1Vertical)
165                     {
166                         // We know one line is vertical so just solve for the
    intersecting point in the second line
167                         px.x = p2.x;
168                         px.y = (m2 * p2.x) + b2;
169                     }
170                     else
171                     {
172                         if(line2Vertical)
173                         {
174                             // We know one line is vertical so just solve for the
    intersecting point in the second line
175                             px.x = p1.x;
176                             px.y = (m1 * p1.x) + b1;
177                         }
178                         else
179                         {
180                             // x = (b2 - b1)/(m1 - m2), y = (m1b2 - m2b1)/(m1 - m2)
181                             if((m1 - m2) == 0) // Avoid Divide by zero
182                             {
183                                 px.x = 0.0f;
```

```
184                             px.y = 0.0f;
185                         }
186                         else
187                         {
188                             px.x = (b2 - b1) / (m1 - m2);
189                             px.y = ((m1 * b2) - (m2 * b1)) / (m1 - m2);
190                         }
191                     }
192                 }
193 
194                 // Find midpoint of p1px and p2px using midpoint formula
195                 // (x,y) = ((x1 + x2)/2, (y1 + y2)/2)
196 
197                 // x = (x1 + x2)/2,  y = (y1 + y2)/2
198                 POINTF p5 = {(p1.x + px.x) / 2, (p1.y + px.y) / 2};
199 
200                 // x = (x1 + x2)/2,  y = (y1 + y2)/2
201                 POINTF p6 = {(p2.x + px.x) / 2, (p2.y + px.y) / 2};
202 
203                 // Find intersection of line perpendicular to p1px passing through p5
204                 // and line perpendicualr to p2px passing through p6
205                 // and then solve the two equations (two equations two unknowns)
206 
207                 // b5 = y - m5x
208                 float b5 = p5.y - (m5 * p5.x);
209 
210                 // b6 = y - m6x
211                 float b6 = p6.y - (m6 * p6.x);
212 
213                 // Now find intersection of the two lines
214                 // This is the center of the cirle that the arc to formed from
215                 // x = (b6 - b5)/(m5 - m6), y = (m5b6 - m6b5)/(m5 - m6)
216                 POINTF px2;
217                 if((m5 - m6) == 0) // Avoid Divide by zero
218                 {
219                     px2.x = 0.0f;
220                     px2.y = 0.0f;
221                 }
222                 else
223                 {
224                     px2.x = (b6 - b5) / (m5 - m6);
225                     px2.y = ((m5 * b6) - (m6 * b5)) / (m5 - m6);
226                 }
227 
228                 // Now find the radius of the circle using the distance formula
229                 // d = sqrt((x2 - x1)^2 + (y2 - y1)^2)
230                 float r = sqrt(pow(px2.x - px.x, 2.0f) + pow(px2.y - px.y, 2.0f));
231 
232                 // Set the arc direction, if the angle between the two points
233                 // is greater than 180degrees then change direction
234                 if((float)((j * fAngle) - (i * fAngle)) > (float)fPI)
235                     SetArcDirection(hdc, AD_COUNTERCLOCKWISE); // AD_CLOCKWISE or AD_COUNTERCLOCKWISE
236                 else
237                     SetArcDirection(hdc, AD_CLOCKWISE); // AD_CLOCKWISE or AD_COUNTERCLOCKWISE
238 
239                 // Draw arc in bounding rectangle with center at px2 and radius r
240                 Arc(hdc,   // handle to device context
241                     (int)(px2.x - r + 0.5f),   // x-coord of rectangle's upper-left corner
242                     (int)(px2.y + r + 0.5f),   // y-coord of rectangle's upper-left corner
243                     (int)(px2.x + r + 0.5f),   // x-coord of rectangle's lower-right corner
244                     (int)(px2.y - r + 0.5f),   // y-coord of rectangle's lower-
```

```
    right corner
                        (int)(p1.x + 0.5f),   // x-coord of first radial ending point
                        (int)(p1.y + 0.5f),   // y-coord of first radial ending point
                        (int)(p2.x + 0.5f),   // x-coord of second radial ending point
                        (int)(p2.y + 0.5f));  // y-coord of second radial ending point

                    //Rectangle(hdc, (int)(px2.x - r), (int)(px2.y + r), (int)(px2.x + r), (int)(px2.y - r));

                    //MoveToEx(hdc, (int)p1.x, (int)p1.y, NULL);
                    //LineTo(hdc, (int)p2.x, (int)p2.y);

                    //MoveToEx(hdc, (int)p3.x, (int)p3.y, NULL);
                    //LineTo(hdc, (int)p4.x, (int)p4.y);

                    //MoveToEx(hdc, (int)p1.x, (int)p1.y, NULL);
                    //LineTo(hdc, (int)p4.x, (int)p4.y);

                    //MoveToEx(hdc, (int)p2.x, (int)p2.y, NULL);
                    //LineTo(hdc, (int)p3.x, (int)p3.y);

                    //MoveToEx(hdc, (int)p5.x, (int)p5.y, NULL);
                    //LineTo(hdc, (int)px2.x, (int)px2.y);

                    //MoveToEx(hdc, (int)p6.x, (int)p6.y, NULL);
                    //LineTo(hdc, (int)px2.x, (int)px2.y);
                }
            }
        }
    }
    SelectObject(hdc, hOldPen);
    DeleteObject(hPen);
}
```

```
1  /*****************************************************
2  * Copyright (c): 2006, All Rights Reserved
3  * Project:       SJSU Masters Project
4  * File:          Nussinov.h
5  * Purpose:       Header file for the Bracketed Graph
6  *
7  * Start Date:    10/6/2006
8  * Programmer:    Brandon Hunter
9  *****************************************************/
10
11 #pragma once
12
13 #include "RNA.h"
14 #include "Nussinov.h"
15
16 class BracketedGraph
17 {
18 private:
19
20 public:
21     BracketedGraph(void);
22     ~BracketedGraph(void);
23
24     void Draw(HDC hdc, RNA *pRNA, Nussinov *pNussinov, float xsize, float ysize);
25 };
```

```
/*****************************************************
* Copyright (c): 2006, All Rights Reserved
* Project:       SJSU Masters Project
* File:          Nussinov.cpp
* Purpose:       Class implementation of the Bracketed Graph
*
* Start Date:    10/6/2006
* Programmer:    Brandon Hunter
*
*****************************************************/

#include "StdAfx.h"
#include "BracketedGraph.h"
#include "math.h" // for the floor function
#include "stdio.h" // for the sprintf function used to convert float to char*

// FUNCTION:  BracketedGraph
// Default Constructor
//
// @param   undefined   void
// @return  void
BracketedGraph::BracketedGraph(void)
{
}

// FUNCTION:  ~BracketedGraph
// Default Destructor
//
// @param   undefined   void
// @return  void
BracketedGraph::~BracketedGraph(void)
{
}

// FUNCTION:  Draw
// This function draws the BracketedGraph bitmap onto the device context
//
// @param   hdc         HDC         handle to a bitmap device context
// @param   *pRNA       RNA         pointer to a RNA class
// @param   *pNussinov  Nussinov    pointer to a Nussinov class
// @param   xsize       float       x dimension, size of bitmap
// @param   ysize       float       y dimension, size of bitmap
// @return  void
void BracketedGraph::Draw(HDC hdc, RNA *pRNA, Nussinov *pNussinov, float xsize, float ysize) {
    float fBorder = 5.0f;
    float fCellSizeX = 20;
    float fCellSizeY = 20;
    float fCellLeft = fBorder;
    float fCellTop = fBorder;
    float xPos = fCellLeft, yPos = fCellTop;
    int RNALength = pRNA->getLength();
    int iCellsPerLine = (int)floor((xsize - (fBorder * 2)) / fCellSizeX); // Number of cells that fit on a line
    int iLineCount = RNALength / iCellsPerLine; // Number of complete lines
    int iRemainder = RNALength % iCellsPerLine; // Number of cells remaining on last line

    COLORREF Bkgrnd;
    COLORREF OldBkgrnd;
    COLORREF Foregrnd;
    COLORREF OldForegrnd;

    // Create the font and the foreground color
    HFONT hFont = CreateFont(15, 0, 0, 0, FW_SEMIBOLD, FALSE, FALSE, FALSE,
    ANSI_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
    FF_DONTCARE, TEXT("Arial"));
```

```
63      HFONT hOldFont = (HFONT)SelectObject(hdc, hFont);
64      OldForegrnd = GetTextColor(hdc); // Get the Original foreground color
65
66      xPos = 15;
67      yPos = 10;
68
69      // Print the Nussinov Algorithm Type
70      SetTextColor(hdc, RGB(0, 0, 255)); // Set the foreground color to Blue
71      TextOutA(hdc, xPos, yPos, "Algorithm Type:", 15);
72      SetTextColor(hdc, RGB(163, 21, 21)); // Set the foreground color to Red
73      switch(pNussinov->getType())
74      {
75      case Nussinov::NussinovType::NussinovStandard:
76          TextOutA(hdc, xPos + 135, yPos, "Nussinov Standard", 17);
77          break;
78      case Nussinov::NussinovType::NussinovSCFG:
79          TextOutA(hdc, xPos + 135, yPos, "Nussinov SCFG", 13);
80          break;
81      default:
82          //  It's not a valid value, raise an exception.  We should never get here...
83          throw "An invalid Nussinov Type has been encountered.";  // Could use
    RaiseException(ERROR);
84          break;
85      }
86
87      // Print the sequence length on the facet
88      yPos += 20;
89      char strNumber[64]; // Buffer for number
90      itoa(RNALength, strNumber, 10);
91      SetTextColor(hdc, RGB(0, 0, 255)); // Set the foreground color to Blue
92      TextOutA(hdc, xPos, yPos, "Sequence Length:", 16);
93      SetTextColor(hdc, RGB(163, 21, 21)); // Set the foreground color to Red
94      TextOutA(hdc, xPos + 135, yPos, strNumber, strlen(strNumber));
95
96      // Print the step position
97      yPos += 20;
98      itoa(pRNA->getStepPosition(), strNumber, 10);
99      SetTextColor(hdc, RGB(0, 0, 255)); // Set the foreground color to Blue
100     TextOutA(hdc, xPos, yPos, "Step Position:", 14);
101     SetTextColor(hdc, RGB(163, 21, 21)); // Set the foreground color to Red
102     TextOutA(hdc, xPos + 135, yPos, strNumber, strlen(strNumber));
103
104     // Print the optimal score on the facet
105     yPos += 20;
106     switch(pNussinov->getType())
107     {
108     case Nussinov::NussinovType::NussinovStandard:
109         itoa(pNussinov->getMatrix()[0][pRNA->getStepPosition() - 1], strNumber, 10);
110         break;
111     case Nussinov::NussinovType::NussinovSCFG:
112         sprintf(strNumber, "%.4f", pNussinov->getCYKMatrix()[0][pRNA->getStepPosition
    () - 1]);
113         break;
114     default:
115         //  It's not a valid value, raise an exception.  We should never get here...
116         throw "An invalid Nussinov Type has been encountered.";  // Could use
    RaiseException(ERROR);
117         break;
118     }
119     SetTextColor(hdc, RGB(0, 0, 255)); // Set the foreground color to Blue
120     TextOutA(hdc, xPos, yPos, "Optimal Score:", 14);
121     SetTextColor(hdc, RGB(163, 21, 21)); // Set the foreground color to Red
122     TextOutA(hdc, xPos + 135, yPos, strNumber, strlen(strNumber));
123
124     // Print the traceback path
125     yPos += 40;
126     SetTextColor(hdc, RGB(0, 0, 255)); // Set the foreground color to Blue
```

```
127     TextOutA(hdc, xPos, yPos, "Traceback Path:", 15);
128     SetTextColor(hdc, RGB(0, 128, 0)); // Set the foreground color to Green
129     TextOutA(hdc, xPos + 5, yPos + 15, "Row:", 4);
130     SetTextColor(hdc, RGB(163, 21, 21)); // Set the foreground color to Red
131     TextOutA(hdc, xPos + 5, yPos + 30, "Col:", 4);
132     List::Node_type *node_ptr = pNussinov->listTraceback.head;
133     int Offset = 60;
134     while(node_ptr != NULL)
135     {
136         char strNumberRow[15];
137         itoa(node_ptr->i, strNumberRow, 10);
138         int lenRow = strlen(strNumberRow);
139         SIZE sizeRectRow;
140         GetTextExtentPointA(hdc, strNumberRow, lenRow, &sizeRectRow);
141 
142         char strNumberCol[15];
143         itoa(node_ptr->j, strNumberCol, 10);
144         int lenCol = strlen(strNumberCol);
145         SIZE sizeRectCol;
146         GetTextExtentPointA(hdc, strNumberCol, lenCol, &sizeRectCol);
147 
148         if(sizeRectRow.cx > sizeRectCol.cx)
149         {
150             if(Offset + sizeRectRow.cx > (xsize - fBorder))
151             {
152                 Offset = 20;
153                 yPos += 35;
154             }
155 
156             SetTextColor(hdc, RGB(0, 128, 0)); // Set the foreground color to Green
157             TextOutA(hdc, Offset, yPos + 15, strNumberRow, lenRow);
158 
159             SetTextColor(hdc, RGB(163, 21, 21)); // Set the foreground color to Red
160             TextOutA(hdc, Offset + (sizeRectRow.cx / 2) - (sizeRectCol.cx / 2), yPos +↙
     30, strNumberCol, lenCol);
161 
162             Offset += sizeRectRow.cx + 5;
163         }
164         else
165         {
166             if(Offset + sizeRectCol.cx > (xsize - fBorder))
167             {
168                 Offset = 20;
169                 yPos += 35;
170             }
171 
172             SetTextColor(hdc, RGB(0, 128, 0)); // Set the foreground color to Green
173             TextOutA(hdc, Offset + (sizeRectCol.cx / 2) - (sizeRectRow.cx / 2), yPos +↙
     15, strNumberRow, lenRow);
174 
175             SetTextColor(hdc, RGB(163, 21, 21)); // Set the foreground color to Red
176             TextOutA(hdc, Offset, yPos + 30, strNumberCol, lenCol);
177 
178             Offset += sizeRectCol.cx + 5;
179         }
180 
181         node_ptr = node_ptr->next;
182     }
183 
184     // Print the bracketed graph
185     yPos += 70;
186     SetTextColor(hdc, RGB(0, 0, 255)); // Set the foreground color to Blue
187     TextOutA(hdc, xPos, yPos, "Bracketed Representation:", 25);
188     //SetTextColor(hdc, RGB(0, 128, 0)); // Set the foreground color to Green
189     //TextOutA(hdc, xPos + 5, yPos + 15, "Row:", 4);
190     //TextOutA(hdc, xPos + 5, yPos + 30, "Col:", 4);
191     SetTextColor(hdc, RGB(163, 21, 21)); // Set the foreground color to Red
```

```
192     Offset = 20;
193     for(int x = 0; x < RNALength; x++)
194     {
195         SIZE sizeRectBase, sizeRectPair;
196 
197         char strBase[2];
198         strBase[0] = pRNA->getSequence()[x];
199         strBase[1] = '\0';
200 
201         if(x >= pRNA->getStepPosition())
202         {
203             SetTextColor(hdc, RGB(0, 128, 0)); // Set the foreground color to Green
204             GetTextExtentPointA(hdc, "-", 1, &sizeRectBase);
205             TextOutA(hdc, Offset, yPos + 15, "-", 1);
206             GetTextExtentPointA(hdc, ".", 1, &sizeRectPair);
207             SetTextColor(hdc, RGB(163, 21, 21)); // Set the foreground color to Red
208             TextOutA(hdc, Offset + (sizeRectBase.cx / 2) - (sizeRectPair.cx / 2), yPos
     + 30, ".", 1);
209         }
210         else
211         {
212             SetTextColor(hdc, RGB(0, 128, 0)); // Set the foreground color to Green
213             GetTextExtentPointA(hdc, strBase, 1, &sizeRectBase);
214             TextOutA(hdc, Offset, yPos + 15, strBase, 1);
215             SetTextColor(hdc, RGB(163, 21, 21)); // Set the foreground color to Red
216             if(pRNA->getPairing()[x] == x)
217             {
218                 GetTextExtentPointA(hdc, "-", 1, &sizeRectPair);
219                 TextOutA(hdc, Offset + (sizeRectBase.cx / 2) - (sizeRectPair.cx / 2),
    yPos + 30, "-", 1);
220             }
221             else
222             {
223                 if(pRNA->getPairing()[x] > x)
224                 {
225                     GetTextExtentPointA(hdc, "(", 1, &sizeRectPair);
226                     TextOutA(hdc, Offset + (sizeRectBase.cx / 2) - (sizeRectPair.cx /
    2), yPos + 30, "(", 1);
227                 }
228                 else
229                 {
230                     GetTextExtentPointA(hdc, ")", 1, &sizeRectPair);
231                     TextOutA(hdc, Offset + (sizeRectBase.cx / 2) - (sizeRectPair.cx /
    2), yPos + 30, ")", 1);
232                 }
233             }
234         }
235         if(Offset > (xsize - 30))
236         {
237             Offset = 20;
238             yPos += 35;
239         }
240         else
241             Offset += 15;
242     }
243 
244 
245     SetTextColor(hdc, OldForegrnd);  // Set the foreground color back to original
    color
246     SelectObject(hdc, hOldFont); // Set the font back to original value
247     DeleteObject(hFont); // delete the font object
248 }
```