Identifying the Molecular Lesions in *zipper* using Variant Calling

Fall Semester 2019: Independent study report

Hector de Galard

**Abstract**

The Kiehart Lab focuses on an important facet of life, cell sheet morphogenesis, a highly conserved process that is required for neural tube development and palate formation. We use the dorsal closure stage of *Drosophila melanogaster* embryogenesis as a model system to study morphogenesis and wound healing in vertebrates*.* During dorsal closure, lateral epidermal sheets progress toward the dorsal midline and cover an opening that is filled with amnioserosa cells that ingress from the cell sheet and apoptose. Many genes are required during dorsal closure, including *zipper*, the locus that encodes the heavy chain of nonmuscle myosin II. Our lab has identified, but not sequence verified, several mutant alleles of *zip* and the surrounding gene region using EMS mutagenesis (1). Here we use whole genome sequencing and computational approaches to identify other mutations in these lesions. In addition, we seek to identify the genomic location of two GAL4 transgenes, one responsible for tissue specific expression in the leading edge of the lateral epidermal cells and the other responsible for expression in the periphery of the amnioserosa. Identifying where these GAL4 knockins are located in the genome will provide information about the genes associated with these unique patterns. I have established the necessary stocks to obtain homozygous *zip* alleles and GAL4 insertions. Genomic DNA was extracted, purified and submitted to the Duke Center for Genomic and Computational Biology for NOVASEQ 6000 whole genome sequencing. The average quality score for the sequences ranged from 34.48 – 35.09, indicating extremely accurate results. This semester’s work will focus primarily on the analysis of these reads using computational approaches to identify the lesions of interest.

**Introduction**

Animal embryogenesis is defined as the process where the embryo forms and the embryonic cells differentiate, resulting in cell sheet morphogenesis, of which dorsal closure is an essential step. Before dorsal closure, a dorsal hole filled with a one cell thick sheet of squamous cells called the amnioserosa (2). During closure, the amnioserosa cells oscillate, ingress and apoptose, thereby providing a force for closure while allowing the two lateral epidermal cell sheets to close to dorsal hole (2). Furthermore, the two lateral epidermal cell sheets contain actomyosin rich purse strings near their leading edge, which also provide mechanistic force for dorsal closure (2). A key component involved in dorsal closure is nonmuscle myosin (myosin II) heavy chain, the product of the *zipper* gene (1). Myosin II’s are important actin-binding, force-producing molecular motors that are composed of two identical heavy chains, and two sets of light chains (1). When homozygous null alleles of nonmuscle myosin are present during dorsal closure, the resulting cell shapes are abnormal, tissue integrity is disrupted and dorsal closure fails indicating a requirement for proper myosin function during morphogenesis.

The introduction of ethylmethane sulfonate (EMS) mutagens has induced mutations identified in a screen by Young et al., (1), generating mutant alleles of zip. Some of the observed phenotypes could be molecularly characterized, but not all have known molecular lesions. One goal of this research is to identify the variants in the lesions of interest in nine *zip* allele fly lines to develop a better understanding of the genotypic changes bringing about the phenotypic changes. These molecular lesions could be in coding, non-coding or regulatory sequences.

A second goal of this project is to identify the genomic locations of two inserted Gal4 transgenes that could potentially identify the genes responsible for interesting expression patterns in the leading edge of the lateral epidermis (LE-GAL4) or the peripheral cells of the amnioserosa (PAS-GAL4) (3). The GAL4-UAS gene expression system (adopted by Brand and Perrimon in 1993 for *Drosophila)* (3), is a bipartite system that includes the Gal4 gene (encoding the yeast transcription activator Gal4) and the upstream activation sequence (UAS) enhancer that GAL4 binds to and activates transcription. The GAL4 transgene has been inserted in a variety of gene regions and a gene of interest inserted downstream of the UAS sequence will be expressed in the pattern of the gene region that the GAL4 is inserted in. The expression pattern generated by two existing Gal4 ‘reporter’ lines, LE-GAL4 and PAS-GAL4, are of interest in the study of dorsal closure because they express in the region of overlap between lateral epidermis and amnioserosa tissues. Although PAS-GAL4 was generated by a fusion of the *escargot* enhancer to Gal4, *escargot* is not expressed at a detectable level in the PAS cells (4), thus identification of the genomic insertion site is key to understanding the basis of this unique expression pattern. Using the known nucleotide sequence of GAL4, whole genome sequencing can identify the exact insertion site of the GAL4 transgene, and thereby provide information about the gene products that may be expressed in this important tissue overlap region.

Here we use whole genome sequencing to identify variants of interest using a variant calling workflow. The sequences were check to ensure they were of high quality. Then, they were aligned to a reference, the alignments were processed to extract variants from individual samples and finally, the samples aligned to the same reference were put in a cohort and variants belonging to that cohort were identified and put in a table. To date, one table has been made with variants of four *zip* alleles which will be analysed to identify the variant causing phenotype. Another set of five zip alleles have been prepared for variant calling, but the step has not yet been performed. Finally, the GAL4 sequences have yet to be processed to identify their genomic locations.

**Materials, Methods and Results**

**Assumptions**

The method presented makes a few assumptions. First of all, it is assumed that the sequences provided are all sequenced in the same PCR lane. This is important because errors in PCR lanes are errors that will be consistent throughout the sequences. If multiple lanes were used, the sequenced reads would have to be identified differently to accommodate for potential errors or consistencies. This is mentioned in the methods, but if different lanes were used, the method would need to be accommodated. Next, the method assumes that the reads are all paired ended, which means reads are sequenced in both directions (5’ to 3’ and 3’ to 5’), and not single ended, where reads are sequenced either starting from 5’ or from 3’ ends. While the overall layout of the method would not change if the reads were single ended, the options used with various tools would need to be accommodated as well because the tools would not need to worry about finding pairs of reads. The last, and most obvious yet crucial assumption, is that the reads are around 120bp long and were created using DNAseq, which sequences both introns and exons, as opposed to RNAseq, which only sequences exons. As much as this is obvious, analysing introns is crucial to understanding gene expression because transcriptional factors are present in introns, and the methods to analyse both are very different.

**GATK Best Practices Workflow**

The Broad Institute created a best practices workflow (Figure 1) that uses their tool, Genome Analysis Toolkit (GATK) for variant calling (5). This workflow is very useful and important to follow for proper variant calling.

**Tools**

In addition to GATK, a variety of tools are required to perform a successful analysis. One important program is FastQC (6), which serves as a quality control. It provides information on the quality statistics from the sequence provided by the facility. Another toolkit that can be useful is BBMap and the shell script BBDuk (7), which is used for trimming and solving any issues with quality. Next, the aligner used to align reads with the desired reference is BWA (8), using the MEM function. PCR duplicates are removed using the MarkDuplicates function from the Picard Tools software suite (9), and manipulation of SAM or BAM files was performed with the SAMTools software suite (10). A proper installation is required to use these tools, and the instructions can be found in the README files that come with the package when installed.

**Methods and Results**

For any project, and a coding one in particular, it is important to remain organized, as different files, programs and scripts will need be called. When these are called, terminal requires the user to input the path to reach them. Therefore, at the beginning of a project, a new folder should be created, and all of the code will be run through this folder. To illustrate this method, a folder called “Hector”, which will simply be in the desktop, is created using this code:

*Cd Desktop; mkdir Hector; cd Hector*

Here, “cd” indicates to terminal that the user wishes to enter this directory, and “mkdir” creates the directory “Hector”. From now on, every time code is run, terminal needs to be in the “Hector” directory and can be done simply by typing “cd Desktop/Hector” in terminal. Furthermore, it is crucial that all sequenced data, applications and packages be somewhere in this directory. Finally, whenever a file is used, its path will also need to be indicated. If a file called “file1” is just in the original “Hector” directory, then the file can be simply be referred to as “file1”. However, if this same file is inside another file called “fileA”, then whenever the file is called, it is important to indicate its location with “fileA/file1”. This will tell terminal where the desired file is. With all the applications mentioned in the materials section installed in the correct folder, the project can begin.

The first step for the analysis of any genomic data is the program FastQC. Normally, this should just be a preventive measure to check for quality as the majority of sequences are accurate, but it can be useful to identify potential errors in per base quality scores, sequence quality scores or adaptor content. If any errors are observed, the shell script bbduk.sh can be used from the package BBMap. This script is capable of removing low quality base pairs, identifying adaptors and removing them, removing X number of bases on the left or the right of each sequence and much more. How this package is used varies case by case, so no default code can be used for every operation. However, in the case of this project, quality control report came back positive from FastQC, alignment could begin without requiring the use of BBMap.

To correctly align the paired-end sequences that we have, a number of steps need to be taken. First, a reference sequence, in .txt or .fasta format, needs to be downloaded either from a provider, or directly from FlyBase if the area of interest is known. Furthermore, this file must be a Plain Text, as opposed to a Rich Text Format, because the aligner does not accept those. The format of the file can easily be changed reformatting it. Let’s call this file “reference.txt”. The next step is to index this reference file using the bwa package and the following code:

*bwa-0.7.17/bwa index reference.txt*

This will create a variety of files that are required later to produce the alignment. The goal of indexing is to save processing power by narrowing down the potential query. A good analogy for indexing is giving pages to a book, it helps the reader find what he is looking for. After indexing the reference, it is also necessary to create a fasta file index using this code:

*samtools-1.9/samtools faidx reference.txt*

This creates an index with single record lines that includes information on each line of the reference file. The goal is the same as above, improve efficiency. Finally, a sequence dictionary describing the contents of the reference file needs to be created:

*java -jar picard.jar CreateSequenceDictionary REFERENCE=reference.txt OUTPUT=reference.dict*

The reference file is now properly configured, and the alignment can be performed using the bwa package. Alignment is rather straight forward. The sample given to us by the sequencing facility contains thousands upon thousands of short reads, usually 120 base pairs, that represent every part of the genome multiple times (depending on the coverage). The aligner uses the reference file as a reference to which it will compare all the different reads for compatibility, and will output a SAM file with only the reads that were compatible to a high degree. There are many different options that can be activated or deactivated to tamper with the sensibility of the aligner, but when the coverage for the sequenced data is high enough, the default values are good enough. If coverage is really low, it might be a good idea to lower the sensitivity to have enough of the reads aligning. On the other hand, if the coverage is very high but there is a lot of mutation noise, it can be helpful to increase the sensitivity to only accept the best reads. Careful not to increase the sensitivity too much, or else the variants that are being looked for can be removed as the program deems them to be incompatible. However, for the default options, this is the code required to align the data:

*bwa-0.7.17/bwa mem reference.txt in1=zip1\_S1.fastq.gz in2=zip1\_S2.fastq.gz > zip1.sam*

The output will be a SAM file, which stands for Sequence Alignment Map, and has alignment information such as the read, the mapping position and the quality. This file will only contain reads relevant to the reference provided. For additional analysis to proceed however, this SAM file needs to be converted to a BAM file, which is just a binary version of a SAM file. This is required for computational reasons and for downstream analysis. The code is as follows:

*samtools-1.9/samtools view -S -b zip1.sam > zip1.bam*

The option “-S” specifies that the input file is a SAM file, while the tag “-b” indicates that the output should be a BAM file. Next, for any meaningful work, the BAM file needs to be sorted. Currently, the file is randomly ordered with respect to their position to the reference genome. To change that, the following code is required:

*samtools-1.9/samtools sort zip1.bam -o zip1\_sorted.bam*

Finally, this new, sorted BAM file needs to be indexed again as to permit downstream analysis:

*samtools-1.9/samtools index zip1\_sorted.bam*

With a complete file ready, read groups need to be added to provide context to the BAM file. A read group is an information header that defines a certain set of reads based on its preparation. Read groups can get very complicated when multiplexing is involved, since subset of reads may come from separate library runs on a different number of lanes for different samples. However, in simple cases where a single library preparation was performed on one lane for different samples, read groups are very straightforward. It is important to get this right however, and this information is crucial for downstream analysis and correct cohort creation. There are five read groups that need to be assigned. The first one is the read group identifier (“RGID”) and is just a unique ID that differentiates your samples from the different lanes. If all the samples came from the same lane, then they can just be differentiated by A, B, C etc. However, if samples were run on different lanes, it would be important to specify that using terms like S1L1 (for sample 1, lane 1), S2L1, S1L2, S2L2. This differentiation permits recalibration of bases during downstream analysis. The next read group is the platform unit (“RGPU”). This read group is again only important for multiplexed data, in which case the lane needs to be marked. However, if the same lane is used for each sample, then the platform unit is just the instrument used, and most likely “NovaSeq”. The next read group is straight forward, as is just the sample name (“RGSM”), and should be the same as the ID. The platform/technology used also needs to be identified (“RGPL”) and is most often illumina. Finally, the last read group to be added is the DNA preparation library identifier (“RGLB”), which is needed by MarkDuplicates in case the same DNA library was sequenced on multiple lanes. If only a single lane was used, this is arbitrary and just needs to be the same throughout. The code to assign read groups is as follows:

*java -jar picard.jar AddOrReplaceReadGroups I=zip1\_sorted.bam O=zip1\_RG.bam RGID=1 RGLB=A RGPL=illumina RGPU=NovaSeq RGSM=1*

The BAM file is nearly ready for variant calling, and one more program needs to be run to improve variant calling quality and give better results. The program is called MarkDuplicates, and is a tool present in the Picard package. MarkDuplicates is essential for genomic analysis as it removes PCR duplicates that arise during replication. Essentially, the program will compare similar reads or areas and group them as being the duplicates of each other (11). Within each group, the best read will be kept as is, and the rest will be marked as duplicates. The importance of this tool comes when a variant table is created, as it will remove possible bias from regions that are over-represented. If MarkDuplicates is not run, errors that occurred during PCR could be amplified to lead to false positives, or areas could be over-represented, and the quality score could be incorrect. The code required to run this tool is the follow:

*java -jar picard.jar MarkDuplicates I=zip1\_RG.bam O=zip1\_MD.bam M=zip1\_metrics.txt*

The output will contain the exact same information as the input, with the only difference being that the output file will have reads that are marked as PCR duplicates, which will be used by the variant calling tool. Furthermore, MarkDuplicates will generate a metrics file (“M=zip1\_metrics.txt) which has information about the number of duplicates and other metrics. Regardless of if the file is important for the project, it is required for the code to run. Following MarkDuplicates, the new BAM file needs to be indexed again, using the same code as before. The file is now ready for the Variant Discovery step of the GATK Best Practices workflow.

The first step towards variant discovery is to run each individual analysis-ready BAM files through a tool called HaplotypeCaller, which calls SNPs and indels via local re-assembly of haplotypes (12). What that means is that it will reassemble regions showing variation to identify variant and assign a quality score to it, along other statistics. The output is an intermediate GVCF file that can be combined with other GVCF files to form a cohort and create a variant table. Looking at files from the same region as cohorts rather than as individual files is extremely useful for analysis as it helps resolve bias issue and gives more confidence at sites that are difficult to read alone. Figure 2 shows the power of this kind of analysis. If sample #1 was analysed alone, it would be hard to tell if this was a site of variation. However, by adding samples, it is possible to either prove that indeed variation is present, or that this site is just a false positive. The code to run HaplotypeCaller is the following:

*java -jar gatk-4.1.4.0/gatk-package-4.1.4.0-local.jar HaplotypeCaller -R reference.txt -I zip1\_MD.bam -O zip1.g.vcf -ERC GVCF*

The code needs to be repeated for each sample that runs under the same reference, which in the example provided would be “reference.txt”. Next, all these different files need to be combined under a single path directory. This is done this way only because the tool GenotypeGVCFs does not accept multiple inputs. To combine these files, GenomicsDBImport can be used as follows:

*java -jar gatk-4.1.4.0/ gatk-package-4.1.4.0-local.jar GenomicsDBImport -V zip1\_MD.bam -V zip2\_MD.bam … --genomics-workspace-path zip -L “reference-interval”*

This program can accept a very large number of input files and just need to specification “-V filename” repeated for it to work. The genomics workspace path will just create a coding path indicating to the tool that the input, which in this case is “zip”, includes all the files inputted above. Finally, the reference interval is important and needs to be exactly the same as the interval provided by the reference.txt file. In the case of zip, the reference would be “2R:24990570..25011965”, which is the chromosome, the branch and the nucleotide positions of the first and last bases in the sequence used as a reference and needs to be the same for the code to run.

Now that a path directory was created, all the different files can be combined to create the variant table that will give the first results. The output will be a VCF file, and can be created following this code:

*java -jar gatk-4.1.4.0/gatk-package-4.1.4.0-local.jar GenotypeGVCFs -R reference.txt -V -gendb://zip -O zip.vcf*

With the creation of this file, the variant table is officially created. To see these results, the VCF file can be opened with excel and will give columns of results that can be analysed by eye (if there are not too many samples and the reference is not too large). The column that is interesting for the scope of variant discovery is the “INFO” column. The column provides information on the allele in question, but the one that is particularly interesting is the Allele Count (“AC”). When that column shows of value of 2, that means that the variant represented by that row is only present in two alleles, indicating it is unique to one sample. From then, the unique variants can be identified and the change in genotype can be identified for the different phenotypes.

The above protocol was applied to the *zip* allele analysis using the genomic region of *zip* as a reference. Four *zip* alleles, more precisely *zip1.6, zip 3.9, zip 3.12* and *zip-14b,* could be narrowed down to that region following analysis presented by Young et al. (1), demonstrating that the variant was part of the gene and was not of a transcription factor outside the genomic location. On the other hand, a series of *zip* alleles intitled *Pey* has been shown, by the same study, that the variant was outside the genomic region. Therefore, the reference sequence used for these included 2kb pairs before and after the genomic region to identify variants in these regions.

Fortunately, variant calling for *zip* alleles did not require any work on the quality of the sequences as they were of high quality and had removed adaptor sequences, which was verified with FastQC. Then, the *zip* alleles were all aligned to the genomic region reference, the resulting SAM files were transformed to BAM files and read groups were added. MarkDuplicates was used and variant discovery could begin after that. The *Pey* alleles stopped at this stage of processing, but the other four *zip* alleles had their variants discovered using HaplotypeCaller and GenotypeGVCFs. This resulted in the creation of a variant table that included all variants from the four alleles analysed. The table is still very large and unpresentable, as every single variant is presented on it. Some of these variants are of low quality and can be discarded, and others are variants observed in all samples, which are not interesting in the search of unique variants. Once the table is more refined and only includes unique variants of high enough quality, it will be attached in the next report.

**Discussion**

By following this workflow, I have achieved multiple goals this semester. First and foremost, I was able to generate a variants table that includes interesting information about possible mutations that could identify lesions of interest of four different zip alleles. Furthermore, I’ve developed my understanding of computational genomics as I needed to research correct approaches, terminology and code to develop the variant table. While this might have taken me an entire semester, what I have learned will allow me to repeat similar experiments and analysis at a much faster rate. Taking the time to understand the reason behind each step was also a challenge, but eventually benefitted me by improving my overall understanding and my troubleshooting capabilities. Finally, this semester’s work has allowed me to create a general method to computational analysis that can be followed by anyone interested in performing a similar procedure. The code itself is not hard, but with the overwhelming amount of information and tools present, it can be difficult to understand what step to perform and when.

The methods and results make the process seem very straightforward and simple. However, that is not the case, and anyone performing genomic analysis at this level will experience difficulties. Errors and warnings are bound to happen, but thankfully, something exists to assist. The package samtools provides a use a tool called ValidateSamFile that decrypts errors and warnings that do not seem to make sense. If a file is causing problems and cannot be inputted into the next operation, here is what to do. First, ValidateSamFile needs to be run in mode summary with this code:

*java -jar picard.jar ValidateSamFile I=zip1.bam MODE=SUMMARY*

The output will be either be “No errors found”, which means nothing is wrong with the BAM file, or a table will be generated, presenting possible errors and warnings. An error is not the same as a warning because an error must be addressed to continue downstream analysis. On the other hand, a warning may sometimes be ignored, depending on the tools used in downstream analysis. If the table presents some errors, then the following code can be used to present them in a more comprehensible form:

*java -jar picard.jar ValidateSamFile I=zip1.bam IGNORE\_WARNINGS=true MODE=VERBOSE*

Mode verbose will explain in words what the errors are, which can be very helpful for troubleshooting cryptic errors. Errors should then be addressed, and ValidateSamFile should be run again to check if that error was really solved. Once all the errors are solved, the warnings can be addressed. To check in verbose what the warnings are, the same code can be run, by using *IGNORE=type* instead of the command *IGNORE\_WARNINGS=true*.

If ValidateSamFile does not help, online forums and threads might have the answer. Every error has an answer, which can be related to code, assumptions made, options used or because the wrong tool is being used. Users post questions and a few moderators answer these questions with great detail. If the error persists and the answer is not online, checking the code again can sometimes solve everything. A mistake I made that took a long time to answer happened at the GenomicsDBImport step, as I had swapped two numbers from my reference interval. This small mistake stopped all progression and was particularly frustrating. Overall, this process is straightforward, but problems can arise very quickly and can seem very hard to solve. Staying focused, motivated and knowing where to find help are the most important factors for projects such as these.

**Future Directions**

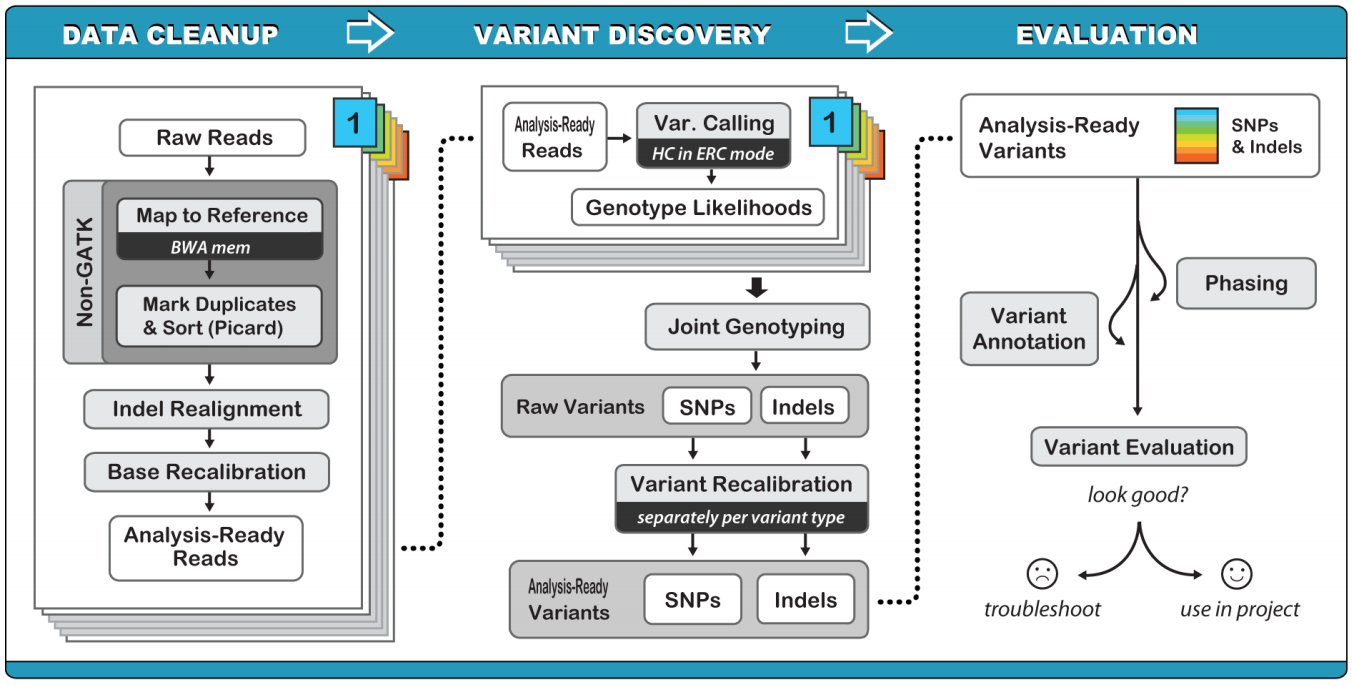
Additional work remains to be completed. First, a deeper understanding of what the VCF table is presenting needs to be developed, because as of right now there are still a lot of things that I don’t understand. Next, and only once a better understanding is developed, the data presented by the variant table will be analysed to determine whether the variants identified are unique and reliable. If it is deemed so, then the allelic variant(s) and location(s) can be recorded. However, if the data is still unclear, then the GATK best practices workflow will be followed. This would include steps such as Variant Recalibration, Phasing and Variant Annotation (Figure 1). These tools’ goal is to improve variant calling quality and provide analysis which would help identify variants. However, considering only four *zip* alleles are being analysed, a small sample size, and that the range of the reference is only that of one gene, it is safe to assume that the current variant table contains all the information necessary to identify the mutations in the lesions of interest.

Furthermore, the variants present in all the samples sequenced should also be recorded, as they can provide useful information for future analysis of samples coming from the same sample pool. For this particular project, one tool, called Base Quality Score Recalibration (BQSR), could unfortunately not be used as one input needs to be known variant sites. Now that a first round of variant calling has been performed, variants present in all samples should be kept and used for BQSR as known variant sites. With the new input of known variants, BQSR looks for systematic patterns and reassigns quality scores based on quality scores of known variants (13). This reassignment improves quality score and can remove false positive results, but can only be run if a known variant pool common to all samples already exists.

Following the analysis of *zip* alleles using the above method, the genomic locations of the two GAL4 transgenes will be analysed, as their insertion locations could provide interesting information about the genes associated with the leading edge epidermal cells and the peripherial amnioserosa cells. However, the method for finding these transgenes would have to be different, as variant calling is not the desired method. Instead, the known nucleotide sequence of GAL4 will act as a reference sequence. While much of the genomic analysis workflow might have parallels, the overall method would not be following the GATK best practices workflow. A new method would have to be created that involves the sequencing of the GAL4 reference sequence to a large reference, such as a whole chromosome or even the whole genome. Once the GAL4 sequence is aligned and finds its match, that location would need to be identified and analysed using FlyBase to check for surrounding genes.

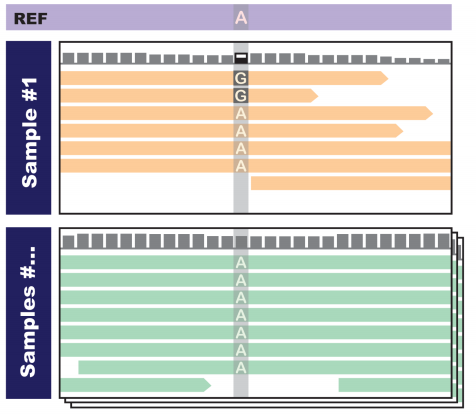
To conclude, four *zip* alleles of interest were fully analysed and a variant table was generated. Furthermore, five other zip alleles have been prepared for variant calling, but the final steps have not been performed yet. This procedure is crucial in understanding the genotypic differences that cause the mutant phenotype. Although the end-goal of a variant table has been generated, understanding how to read the variant table is crucial for discovery of the molecular lesions. Furthermore, additional research has to be performed to develop a procedure to discover the locations of the GAL4 transgenes which could identify other genes of interest. Nevertheless, the overall method appears to be successful and a detailed workflow that others can follow has been created.

**Figures**

Figure 1: GATK Best Practices Workflow

*https://www.broadinstitute.org/partnerships/education/broade/best-practices-variant-calling-gatk-1*

Figure 2: Power of cohort-analysis



*By analysing sample 1 alone, little information is known about that location. By adding multiple samples, it can be deduced that the samples mostly show nucleotide A, and is not a site of possible variation with nucleotide G*

*https://www.broadinstitute.org/partnerships/education/broade/best-practices-variant-calling-gatk-1*

**References**

1. Young et al. 1993. Morphogenesis in *Drosophila* requires nonmuscle myosin heavy chain function. Genes & Dev 7:29-41
2. Kiehart et al. 2017. Cell Sheet Morphogenesis: Dorsal Closure in *Drosophila melanogaster* as a Model System. Annu Rev Cell Dev Biol 33:169-202
3. Brand A. H, Perrimon N. 1993. Targeted gene expression as a means of altering cell fates and generating dominant phenotypes. Development. 118(2):401-415
4. Wada et al. 2007. Specialized extraembryonic cells connect embryonic and extraembryonic epidermis in response to Dpp during dorsal closure in *Drosophila*. Dev Bio 301:340-349
5. https://www.broadinstitute.org/partnerships/education/broade/best-practices-variant-calling-gatk-1
6. https://www.bioinformatics.babraham.ac.uk/projects/fastqc/
7. https://jgi.doe.gov/data-and-tools/bbtools/bb-tools-user-guide/bbduk-guide/
8. Li H. and Durbin R. 2009. Fast and accurate short read alignment with Burrows-Wheeler Transform. Bioinformatics, 25:1754-60
9. http://broadinstitute.github.io/picard
10. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R. and 1000 Genome Project Data Processing Subgroup. 2009. The Sequence alignment/map (SAM) format and SAMtools. Bioinformatics, 25, 2078-9.
11. https://www.biostars.org/p/18784/
12. https://software.broadinstitute.org/gatk/documentation/tooldocs/4.0.8.0/org\_broadinstitute\_hellbender\_tools\_walkers\_haplotypecaller\_HaplotypeCaller.php
13. https://gatkforums.broadinstitute.org/gatk/discussion/44/base-quality-score-recalibration-bqsr