A 3D Visualization of White Blood Cell Populations

A Program to Visualize Analyzed Data from a Blood Sample

ANDREA LUNDKVIST

GABRIELLA NORMAN
A 3D Visualization of White Blood Cell Populations
A Program to Visualize Analyzed Data from a Blood Sample
En 3D visualisering av vita blodkroppspopulationer
Ett program för att visualisera analyserad data från blodprov

Andrea Lundkvist
Gabriella Norman

Degree project in medical engineering
First level, 15 hp
Supervisor at KTH: Mattias Mårtensson & Tobias Nyberg
Examiner: Mats Nilsson

School of Engineering Sciences in Chemistry, Biotechnology and Health
KTH Royal Institute of Technology
SE-141 86 Flemingsberg, Sweden
http://www.kth.se/cbh
2018
Abstract

The goal of this project was to write a program to plot populations of white blood cell types in three dimensions and reduce the information from four dimensions to three in order to make it possible to plot. The dimension reduction should be done with as little loss of information as possible. PCA, principal component analysis, was used to perform the dimension reduction. A program for this was written in MATLAB.

The program that builds the plot of the populations was written in c# using Visual Studios. The visualization was made in Unity. It was possible to rotate and zoom into the plot to get a better view.

The project was given by Boule Medical AB as a part of the development of a new product. Boule Medical AB only had 2D plots and requested a 3D plot in their new product. The program was handed over to Boule Medical AB for future development.

Key words: PCA, dimension reduction, white blood cells, blood analysis, 3D visualization
Sammanfattning

Projektets mål var att skriva ett program som plottar populationer av vita blodkroppar i tre dimensioner samt reducerar information om de vita blodkopparna från fyra dimensioner till tre med så liten förlust av data som möjligt.

PCA, *principal component analysis*, användes för dimensionsreduktionen och ett program för att utföra denna skrevs i MATLAB.


*Boule Medical AB* var uppdragsgivare och det här projektet är en del i deras utveckling av en ny produkt. *Boule Medical* hade bara 2D grafer och efterfrågade en 3D graf till sin nya produkt. Programmet överlämnades till *Boule Medical* för fortsatt utveckling.
Contents

1 Introduction 1
   1.1 Aim .................................................. 2
   1.2 Limitations ......................................... 2

2 Background 3
   2.1 White blood cells .................................... 3
      2.1.1 Neutrophils granulocytes .................... 4
      2.1.2 Eosinophils granulocytes .................. 4
      2.1.3 Basophils granulocytes ..................... 5
      2.1.4 Monocytes ...................................... 5
      2.1.5 Lymphocytes .................................. 5
   2.2 Flow Cytometry ....................................... 5
      2.2.1 Analysis of the data ......................... 6
   2.3 Dimension reduction with PCA .................... 6
   2.4 Prototype from Boule Medical AB ................. 7

3 Method 9
   3.1 Dimension Reduction ................................ 9
   3.2 Visualization ....................................... 10

4 Results 11
   4.1 PCA .................................................. 11
   4.2 Program ............................................. 11
      4.2.1 Plot of Original Data ....................... 12
      4.2.2 Plot of Dimension reduction ............... 13

5 Discussion 16
   5.1 Software and Programming choices ............... 16
   5.2 Resulting program .................................. 16
   5.3 Future work ........................................ 17

6 Conclusion 19

References 20

Appendices
   Appendix 1: PCA
   Appendix 2: Import
   Appendix 3: OpenFile
   Appendix 4: CreateSphere
   Appendix 5: Zoom
1 Introduction

White blood cells are a part of the immune system, protecting the body from threats and intruders such as virus and bacteria. Information about the patients’ well-being is gained by analyzing the cells in the blood. The white blood cells can e.g. tell if there is an infection in the body or if something is harming the immune system [1].

Biomedical analysts need to get an overview of the white blood cell population to quickly diagnose abnormalities in a patient’s immune system. The overview is often presented as a plot [2].

This project is executed for Boule Medical AB. Boule Medical AB is a Swedish company that develops, produces and sells blood analysis systems all over the world [3]. Today, at Boule Medical AB, the visualizations of white blood cells are made in 2D, as in Figure 1 below.

![Figure 1: How the plot in the Blood Analysis Machines at Boule Medical AB is visualized today](image)

In this project, the data describing the white blood cells is provided by a blood analysis machine. This data has four parameters representing four measurement angles in the machine. Unfortunately, the plot in Figure 1 is only presenting two dimensions, so the biomedical analysts do not get a full overview. Given the nature of the data, a 4D plot would be the most satisfying. However, since that is not possible, a 3D plot is the best alternative. A reduction of the dimensions can make the four-dimensional data fit into the 3D plot with some loss of information. This way, the biomedical analysts can get a fuller overview of the white blood cell populations.
1.1 Aim

The aim of this project is to create a program that can both visualize parameters for white blood cell populations in 3D and reduce data from four dimensions to three dimensions. Important features of the program are:

- The ability to choose a file.
- The option to visualize dimension-reduced data or to visualize three out of four original parameters.
- Plotted data colored depending on categorization of blood cells.
- The ability to remove debris from the visualization.
- The ability to rotate and zoom.

1.2 Limitations

The program should be written in C# using Microsoft Visual Studios, for Boule Medical AB to be able to use the program afterward.
2 Background

A blood sample is needed to do a blood analysis. The most common blood sample is drawn from the arm and contains venous anoxic blood. A blood analysis is useful in many areas of medical care. It can e.g. determine renal function or infections in the body [4]. Therefore a blood analysis is one of the most common diagnostic methods in the world. Every year 1.2 billion blood analyses are performed around the world, according to F. Dalborg, Boule Diagnostics AB. A complete blood count (CBC) is often executed when analyzing blood. The process to analyze blood can be divided into three steps:

1. Collect the blood sample.
2. Run the sample through a blood analysis machine that performs the CBC.
3. Use the result as a base for diagnosis.

A blood analysis machine counts the blood cells and divides them into red blood cells, platelets, and white blood cells. The amount of each type of blood cell in the sample provides information used for diagnosing [5].

2.1 White blood cells

The white blood cells are a part of the immune system. The task of the immune system is to protect the body from damage. The task of the white blood cells is to incapacitate threats that already entered the body, e.g. virus and bacteria. The white blood cells are divided into groups depending on their tasks and characteristics [6].

All blood cells develop from pluripotent hematopoietic stem cells that exist in the bone marrow. The developmental process of these cells is called hematopoiesis and mainly occurs in the bone marrow. The hematopoietic stem cells can mature into three different stages called the erythroid, myeloid and lymphoid lineages. From these cells, other cells develop. These are called neutrophils granulocytes, eosinophils granulocytes, basophils granulocytes, monocytes, and lymphocytes [1].

<table>
<thead>
<tr>
<th>Cell type</th>
<th>Proportion of leukocytes (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neutrophil</td>
<td>40 - 75</td>
</tr>
<tr>
<td>Eosinophil</td>
<td>1 - 6</td>
</tr>
<tr>
<td>Basophil</td>
<td>&lt;1</td>
</tr>
<tr>
<td>Monocyte</td>
<td>2-10</td>
</tr>
<tr>
<td>Lymphocyte</td>
<td>20-50</td>
</tr>
</tbody>
</table>

Table 1: Distribution of white blood cell types in the human blood [1]
The different white blood cell types have different tasks and characteristics. This means that different cells increase in number depending on what threatens the body. By knowing how many white blood cells of each type a patient has, a physician can determine what kind of threat the patient’s immune system is fighting \[1\]. To determine what type of white blood cell a cell is, their constitution is relevant. An image of the white blood cells can be found in Figure 2 \[8\].

### 2.1.1 Neutrophils granulocytes

Neutrophils are so-called effector cells, which means they actively react to stimuli. Neutrophils fight infections. Large amounts of neutrophils are stored in the bone marrow as reserves to fight infections as effectively as possible. When an infection affects the body, neutrophils are released from the bone marrow and immediately travel to the infected area and enter the damaged tissue. Neutrophils can work under anaerobic conditions, e.g., in damaged tissue. After interacting with the damaged tissue, the neutrophils die. The dead cells form a thick fluid called pus. The pus is eventually decomposed by macrophages \[1\].

Fully mature neutrophils have polymorphonuclears, which is a characteristic feature that is clearly visible in Figure 2. Granules is a nontransparent cluster of different substances depending on cell type. In neutrophils, the cytoplasm contains fine specific granules. The granules vary in size, density, and placement \[9\].

### 2.1.2 Eosinophils granulocytes

Eosinophils’ task is to protect the body from different parasites, e.g., helminth worms. Only a small part of the eosinophils circulates in the blood, the rest rests in tissue \[1\].
The eosinophils’ characteristic appearance mainly depends on the homogeneous granules. They are large and dense. The eosinophils also contain a large number of mitochondria and fine nuclear chromatin [9].

2.1.3 Basophils granulocytes

As seen in Table 1, basophils are the least common white blood cells, and therefore their part in the immune system is not as known as the other white blood cells [1]. Basophils contain a lot of histamines. Almost all of the histamine measured in the blood are associated with basophils [10].

The basophils have large granules. These are purplish-black and usually fill the cytoplasm, as can be seen in Figure 2. The large granules often obscure the nucleus [9].

2.1.4 Monocytes

Monocytes circulate in the blood. When they find infected tissue they travel into the tissue and mature to macrophages. The macrophages’ task is to clean up the tissue by eating dead cells and invading microorganisms [1].

Monocytes are bigger than granulocytes. The cytoplasm has a gray color and is blurry due to containing fine dust of granules. Their nucleus is shaped like a horseshoe which can be seen in Figure 2 [9].

2.1.5 Lymphocytes

Lymphocytes are divided into two groups. One group circulates in the blood and is responsible for the adaptive immune response. The other group is responsible for killing virus-infected cells [1].

The lymphocytes that circulate in the blood are generally small and have a bilaminar cytoplasmic membrane. The nucleus is covered by a double membrane [9].

2.2 Flow Cytometry

To detect and sort the different cell types, a method called Flow Cytometry is used. The blood cells flow through a cylinder one by one. To ensure that not more than one cell at a time flows through the cylinder the cells are surrounded by a cell-free buffer [2].

The cells are irradiated with a laser beam in a cuvette and the scatter is measured for each cell. The laser beam consists of photons and there is an expected path for each photon in the laser beam as long as they do not interact with their surroundings [8]. If a photon does collide with an object in the surroundings there is an interaction between the photon and the object. This makes the path of the photon change, e.g change direction. The interaction is different depending on the object the photon collides with. In the laser beam there are multiple photons that interact and when all this data is collected, information about the objects physiognomy is given. This information is used to identify the object with which the photon interacted [11].
The different types of blood cells are structured differently and the interaction between
the photons in the laser will, therefore, be different. To measure the photons paths, the
light scatter is detected at different angles around the cuvette, see Figure 3 [8].

![Flow Cytometry setup](image)

**Figure 3:** Example setup for the Flow Cytometry method. Figure created by authors

### 2.2.1 Analyzis of the data

The data collected by the Flow Cytometry is analyzed and categorized based on how much
light is scattered in different angles. This is possible since the different blood cells have
different structures, e.g., a blood cell with a nucleus absorbs more light than one without
one and this is seen in the forward scatter. Another example is eosinophils scattering in
a wider angle and with more depolarized scatter than neutrophils.

It is not always easy to differ the cell types from each other. A damaged cell of one type
can make it look similar to a completely different cell type, and the differences between
cell types are not always that big [12].

### 2.3 Dimension reduction with PCA

When data has more parameters than manageable, a dimension reduction is necessary. A
popular method for this is principal component analysis, PCA. PCA detects the vectors
with highest variance in data [13]. To be able to perform PCA five requirements need to
be met

1. The data must contain multiple variables collected at a continuous level.
2. The variables must have a linear relationship.
3. The size of the data must be big enough, a minimum of 150 data points is recommended.

4. The variables must have adequate correlation.

5. The data should not include any significant outliers [14].

The steps of the method are:

1. Calculate the covariance matrix of the data.

2. Calculate the eigenvalues and eigenvectors of the covariance matrix.

3. Choose the new bases. The eigenvector with the biggest eigenvalue will be the vector with the highest variance. Choose as many bases as needed from the eigenvectors, based on eigenvalues in descending order.

4. Do a base change to the new bases. The data will now be represented by the newfound bases.

Some loss of information will occur but by using the vectors with the highest variance the loss is minimized. PCA is known to be the best method with the mean squared error in mind. On the other hand, PCA is not a good method if the parameters are important for analyzing the data, since the base change makes the data points lose their connection to the original bases. In Figure 4 an example of PCA performed from a 3D space to a 2D space can be seen [13].

![Figure 4](image)

**Figure 4:** Example of PCA reduction from 3D to 2D and how it can be seen. From [15] with permission.

### 2.4 Prototype from *Boule Medical AB*

Bengt-Olof Swing at *Boule Medical AB* informed us about the machine used in this project. The light is measured at four different angles, more precisely the shadow behind the blood cell and three more angles. These four angles are the parameters used for the
axis of the visualization. In one sample around 5000 blood cells are analyzed one by one with the Flow Cytometry technique. The machine is not yet ready for the market and in this project, it is called the prototype.

Bengt-Olof Swing also described his program for categorization. In this program each shatter is categorized. The program has known values for the different cell types. The categories are neutrophil, eosinophil, basophil, monocyte, lymphocyte, debris and undefined. A categorization is made based on the scatter of each cell. The category undefined is set when the scatter can not be categorized to any of the known cell types but probably belongs to a cell. The category debris is set if the scatter does not look like a cell at all. The data set with the scattering values from all four angles and the category are then saved together in an XML-file (Bengt-Olof Swing, Senior System Engineer at Boule Medical AB, 180403).
3 Method

The project was divided into two parts, the dimension reduction and the visualization. Programs were made to do these two parts. More specific features of the different parts are describes below. XML-files was handed over from Boule Medical AB along with a C# class, PulseXmlReader, written by Bengt-Olof Swing at Boule Medical AB for reading the files. An XML-file contains information from one blood sample. The following information is found in an XML-file:

- How many white blood cells that were in the sample.
- Values of the four angles for each cell.
- The categorization of each cell.

3.1 Dimension Reduction

A dimension reduction was made using PCA. As described in the background there are five requirements for the data that need to be met to be able to use PCA. The data in this project meets the requirements 1-4 automatically since there is multiple variables with adequate correlation collected in a sample, there is a relationship between the different cells in one sample and as mentioned there is about 5000 cells, data points, in one data set, which is more than 150. By removing the categories debris and undefined requirement 5 is met, since the categorization is made and these two categories are the outliers. The program was written in MATLAB R2017a 9.0.0 (MathWorksInc., Natick, MA, USA). An XML-file was read and a PCA was performed as described in the background. The new values were listed in a new XML-file.

The MATLAB-function PCA has six steps. They are the following:

2. Removes data marked as debris or undefined.
3. Calculates new bases.
4. Calculates the new values for the data.
5. Writes the data to an XML-file.
6. Writes the bases to a text file.

In the new bases, the data no longer represents the scatter at the different angles. When this is done, it is no longer possible to evaluate the data according to the axis and how large the scatter is in any direction but it is still possible to look at the ratio between the different types.
3.2 Visualization

Microsoft Visual Studios 1.15 (Redmond, WA, USA) with Unity 2017.3.1f1 (San Francisco, CA, USA) was used to access a 3D environment. Unity was used to developed a 3D environment.

In the 3D environment the user was able to choose which file to use and if a dimension reduction should be done. If not, the original file, where no manipulation is made after the PulseXmlReader is performed, can be displayed. If the original was selected, the user could select which three angles out of the four stored in the XML-file should be visualized. Points were colored differently according to the categorization in the XML-file.

The plot was created so it was able to rotate and zoom. It was also possible to choose which types of cells should be shown.

In Microsoft Visual Studio the following libraries were used to create the program:

- UnityEngine
- UnityEngine.UI
- UnityEditor
- System
- System.Linq
- System.Collections.ObjectModel
- System.IO.
4 Results

In this section, the results are presented.

4.1 PCA

The program written for the PCA was made in MATLAB. The program reduces the information given in the original file from 4D to 3D and saves this in a new XML-file. The new bases are also stored. The dimension reduction is made in six functions, with different tasks, seen in Table 2.

The input parameters needed in the program is the name of the file, the dimension reduction will be performed at, i.e the file read from, the name of the new file where the new values for each cell is stored, and an empty txt-file where the new bases are stored.

Table 2: The functions for the PCA made in MATLAB and their aim

<table>
<thead>
<tr>
<th>Function</th>
<th>Aim</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCA</td>
<td>Main function, runs the other functions</td>
</tr>
<tr>
<td></td>
<td>and writes the new bases to the txt-file</td>
</tr>
<tr>
<td>LoadFile</td>
<td>Reads the file with the information about the cells,</td>
</tr>
<tr>
<td></td>
<td>the XML-file created in PulseXmlReader</td>
</tr>
<tr>
<td>RemoveUndef</td>
<td>Removes the pulses categorized as undefined</td>
</tr>
<tr>
<td>RemoveDebris</td>
<td>Removes the pulses categorized as debris</td>
</tr>
<tr>
<td>ReduceDim</td>
<td>Performs the actual PCA, reduce the dimensions</td>
</tr>
<tr>
<td>CreateStructure</td>
<td>Writes the data reduced information</td>
</tr>
<tr>
<td></td>
<td>to the new XML-file</td>
</tr>
</tbody>
</table>

It takes between 20 seconds and five minutes to run a file through the PCA on a regular computer. This depends on the amount of data in the file, which depends on the number of cells in the blood sample analyzed. See Appendix 1 for full code.

4.2 Program

The program that the user will execute is the visualization program created in Microsoft Visual Studio with Unity. The PCA will be executed from within that script.

The structure of the menu is shown in Figure 5. The user can choose a file, whether the original data or PCA-recalculated data should be plotted and if the original data is chosen; three out of four channels. Channels represents the angles. If the PCA-recalculated data is chosen, the MATLAB-script is executed from within the program.
Figure 5: Description of choices to make in the program before the visualization

The visualization program has the following features:

- The ability to choose a file.
- The option to do dimension reduction or choose three of the original parameters to visualize the data.
- Plotted data colored depending on categorization of white blood cell.
- The ability to remove debris from the visualization.
- The ability to rotate and zoom.

To describe the code, a description of the different classes made in Microsoft Visual Studio can be found in Table 3 below.

Table 3: Class Structure of the classes created for visualization in Microsoft Visual Studio

<table>
<thead>
<tr>
<th>Class</th>
<th>Aim</th>
<th>Appendix #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import</td>
<td>To choose the file</td>
<td>2</td>
</tr>
<tr>
<td>OpenFile</td>
<td>Finds the file selected in</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Import and reads it</td>
<td></td>
</tr>
<tr>
<td>CreateSphere</td>
<td>Creates the graphical scene in terms of</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>the plot, buttons, and information</td>
<td></td>
</tr>
<tr>
<td>Zoom</td>
<td>Make the graphical zooming</td>
<td>5</td>
</tr>
</tbody>
</table>

4.2.1 Plot of Original Data

The plot of a data sample can be found in Figure 6 and Figure 7. On the left side, there are buttons for removing debris and other cell types if the user only wants to look at one type. There is also a reload button for rebuilding the plot. The plot is rotated by
click-and-drag of the mouse and zoomed in by right-clicking.

**Figure 6:** Original data where three angles are chosen, plotted in the program from the front

**Figure 7:** Original data where three angles are chosen, plotted in the program when rotated

### 4.2.2 Plot of Dimension reduction

The plot of a data sample recalculated with PCA can be found in Figure 8 and Figure 9. At the top right corner, the new bases are presented. This makes it possible for the user to understand how the data was changed when the dimensions were reduced.
Figure 8: The data after dimension reduction plotted in the program, from the front

Figure 9: The data after dimension reduction plotted in the program, rotated

In Figure 10 and Figure 11 an overview of the difference between plotting 3 channels for the original data or plotting the data after dimension reduction is shown.
Figure 10: Example of plot of original data, where three angels are chosen

Figure 11: Example of plot after dimension reduction
5 Discussion

The discussion is divided into different sections based on the different parts of the project and what could be done in the future.

5.1 Software and Programming choices

The limitation of Microsoft Visual Studios together with our lack of experience with the syntax resulted in a programming structure that is not optimal to follow or understand for others. This can e.g. be seen in the choices of classes made, seen in Table 3. This problem would not occur if we had more experience of working with Microsoft Visual Studios, e.g. if we had more time to learn how Microsoft Visual Studios worked before we started the project. The incomprehensible code may affect the implementation of this program into the prototype negatively.

We could have taken a different approach to this problem. It would have been possible for us to build a 3D environment using only Microsoft Visual Studios. That would have made the start up period longer and it would have been necessary to develop basic 3D features, which takes a long time. This was not possible because of the limited time for this project. If Microsoft Visual Studios only was used the program would probably be slower and the animations would not be smooth. We are also not Microsoft Visual Studio developers, so if we had created features similar to the ones in Unity they would not be as efficient as the ones in Unity.

Many different methods can be used to perform dimension reduction. PCA is, as mentioned in the background, a good method if information loss needs to be minimized and not a good method if the parameters are important. For blood analysis, the parameters are important when doing categorization but when visualizing a blood sample the ratio between the cells are the important aspect. PCA preserves as much as possible of the ratios by minimizing the information loss. Therefore PCA is a good method for this project.

5.2 Resulting program

In Figure 10 and Figure 11 the difference between the original plot and the plot with dimension reduction is shown. By looking at multiple plots, we determined that the separation of the plotted data is better seen in the dimension-reduced data compared to the data visualizing three out of the four channels. When plotting the dimension-reduced data, each cell type groups together and does not mix with other categories as much as when plotting the original data. This could make the analysis of the data easier for the user. It also shows that the dimension reduction is a good feature that could be relevant to implement in a new product.

When the PCA is performed information is lost. The amount of information lost is not calculated in the program but could have a value for the user. To calculate a mean squared error is not hard to do and does not require much performance and therefore not much extra time in the PCA analysis. This could be a continuation of the project if it is of interest to the user.
When performing a PCA reduction on our computers, completing the analysis took longer than desired. The prototype is not expected to have processors as powerful as the ones in our computers, and thus the process would likely take even longer when being performed by the prototype. This can be problematic when used at hospitals because there is often a time pressure for the physicians and they want the result as fast as possible to the patient. If each test takes longer time, one machine can not perform as many test per day which can slow down the process for the physicians.

The program has a clean layout with a simple menu and some options. The options in the menu, seen in Figure 5, are easy to understand if the user knows the purpose of the program. A manual or description of the program would be convenient for new users. After building the plot, six buttons for removing categories occur, seen in Figure 6 and Figure 7. These buttons are labeled with the category they represent and are color coordinated with the color of the category. Therefore the user can easily understand which color represents which category in the plot and also know which button to click for removal.

The requested features of the program were implemented and are working as planned. The features were not requested from an end user. Instead, we and engineers at Boule Medical AB discussed what features would be requested by an end user. The ability to rotate and remove some of the cell types, which we believe is relevant for the end user are seen in figure 7. Another feature is that the new bases expressed in the old bases are displayed in the upper right corner, seen in Figure 8 and Figure 9. For the end user, this information is probably not useful in itself. An information-box where the user can get information about the percentage distribution of the old bases in the new bases might be better, together with an explanation of the reduction. For future development, a reference group of end users would be preferable.

Boule Medical AB gave us this project because they wanted to know if a visualization program would give more value to their product. They let us develop a prototype program for the visualization so they could show it to other parts of the company in order to get approval to start a larger project to develop an embedded program. They also wanted to investigate the possibility of doing a dimension reduction and gave us the task to develop a PCA function. Our project is the first step to develop a program that will give Boule Medical AB a feature to their product. In the future, the program can be used for both quick diagnosis and in research. Researchers can e.g. compare samples from the same patient over a period of time to find patterns in plot depending on the patient’s wellbeing.

5.3 Future work

As previously mentioned, the processor in the prototype is not as powerful as in the computers used in the project. Therefore, the program needs to be more efficient and must not require as much performance from the processor. One way of doing this is to investigate if it is possible to always use the same new bases when performing dimension reduction. This would be possible if the data from healthy samples would always have the same shape. If that is the case the mathematic formulas would decrease rapidly and a PCA calculation would not be necessary. The function ReduceDim in the PCA function,
seen in Table $^2$ would not be used.

Always using the same bases would give the user the possibility to compare dimension-reduced plots with each other. Now the PCA will find new bases for every plot and different samples will have different bases and can therefore not be compared. To find bases that can be used for all future plots, the dimension-reduced bases for many healthy samples need to be compared to each other. From these bases the most suitable bases can be calculated.
6 Conclusion

A program that can visualize white blood cell populations was created fulfilling all the project aims. The user of the program is able to choose file and if the PCA should be performed or not and the plotted data is colored depending on categorization and is both able to zoom and rotate. The program is not as efficient as it needs to be to be implemented into the blood analysis machine under development.
References


Appendices

Appendix 1: PCA

```matlab
function [ S ] = PCA( filename , newname , basisfile )

name = filename;

[ nrofpulses , amps , types , DOMnode ] = LoadFile(name);

[ nrofpulsesU , ampsU , typesU ] = RemoveUndef(nrofpulses , amps , types);

[ nrofpulsesD , ampsD , typesD ] = RemoveDebris(nrofpulsesU , ampsU , typesU);

[ newamps , v1 , v2 , v3 ] = ReduceDim(ampsD , amps , nrofpulses);

newamps = round(newamps);

newDOMnode = CreateStructure(DOMnode, newamps);

s1 = strcat('[' , num2str(v1(1)) , ', ' , num2str(v1(2)) , ', ' , num2str(v1(3)) , ', ' , num2str(v1(4)) , ']' );

s2 = strcat('[' , num2str(v2(1)) , ', ' , num2str(v2(2)) , ', ' , num2str(v2(3)) , ', ' , num2str(v2(4)) , ']' );

s3 = strcat('[' , num2str(v3(1)) , ', ' , num2str(v3(2)) , ', ' , num2str(v3(3)) , ', ' , num2str(v3(4)) , ']' );

disp(s1);
disp(s3);
disp(s3);

fileID = fopen(basisfile , 'w');
fprintf(fileID , s1 , '
');
fprintf(fileID , s2 , '
');
fprintf(fileID , s3 , '
');
fclose(fileID);

S = strcat(s1,s2,s3);

xmlwrite(newname,newDOMnode);
end
```
function [ nrofpulses, amps, types, DOMnode ] = LoadFile( name )

%UNTITLED3 Summary of this function goes here
% Detailed explanation goes here

DOMnode = xmlread(name);

sheet = DOMnode.getFirstChild;
sib = sheet.getNextSibling;

node = sib.getFirstChild;

while ~isempty(node)
    if strcmpi(node.getNodeName, 'nrofpulses')
        nrofpulses = str2double(node.getTextContent);
        break;
    end
    node = node.getNextSibling;
end

node = sib.getFirstChild;

amps = zeros(4, nrofpulses);
place2 = 1;

while ~isempty(node)
    if strcmpi(node.getNodeName, 'pulses')
        pulse = node.firstChild;

        while ~isempty(pulse)
            if strcmpi(pulse.getNodeName, 'pulse')
                pul = pulse.firstChild;

                while ~isempty(pul)
                    if strcmpi(pul.getNodeName, 'Type')
                        types{place2} = char(pul.getTextContent)
                    end
                    if strcmpi(pul.getNodeName, 'amplitudes')
                        amp = pul.firstChild;
                    end
                end
            end
        end
    end
end
place = 1;
while ~isempty(amp)
    if strcmpi(amp.getName, 'amplitude')
        amps(place, place2) = str2double(amp.getTextContent);
        place = place + 1;
    end
    if (place == 5)
        break;
    end
    amp = amp.getNextSibling;
end
place2 = place2 + 1;
end
pul = pul.getNextSibling;
end
end
pulse = pulse.getNextSibling;
end
end
node = node.getNextSibling;
end
amps = amps';
end

function [ nrofpulses, amps, types ] = RemoveUndef( nrofpulses, amps, types )
%UNTITLED3 Summary of this function goes here
% Detailed explanation goes here

index = find(strcmp(types, 'Undef'));
if (index == 0)
    return;
else
    for ii = 1:length(index)
        types{ii} = [];
    end
end
amps(ii,:) = [];
end
nrofpulses = nrofpulses - index;
end
end

function [nrofpulses, amps, types] = RemoveDebris(nrofpulses, amps, types)
% %UNTITLED9 Summary of this function goes here
% Detailed explanation goes here
index = find(strcmpi(types, 'Debris'));
if (isempty(index))
    return;
else
    for ii = 1:length(index)
        types{ii} = [];
        amps(ii,:) = [];
    end
    nrofpulses = nrofpulses - length(index);
end
end

function [newamp, v1, v2, v3] = ReduceDim(amp, Samp, nrofpulses)
% %UNTITLED5 Summary of this function goes here
% Detailed explanation goes here
COV = cov(amp);
[V,D] = eig(COV);
D2 = diag(sort(diag(D),'descend'));
[c, ind] = sort(diag(D),'descend');
V = V(:, ind);
v1 = V(:, 1);
v2 = V(:, 2);
v3 = V(:, 3);
V = \[ v1 \ v2 \ v3 \];

newamp = zeros(3, Snrofpulses);

for \( i = 1: \text{Snrofpulses} \)
    newamp(:, i) = V(Samp(:, i)');
end

defunction [ DOMnode ] = CreateStructure( DOMnode, newamp)

% UNTITLED7 Summary of this function goes here
% Detailed explanation goes here

sheet = DOMnode.getFirstChild;
sib = sheet.getNextSibling;
node1 = sib.getFirstChild;

place2 = 1;
while \( \text{isempty}(\text{node1}) \)
    if strcmpi(node1.getNodeName, 'pulses')
        pulse = node1.getFirstChild;
        while \( \text{isempty}(\text{pulse}) \)
            if strcmpi(pulse.getNodeName, 'pulse')
                pul = pulse.getFirstChild;
                while \( \text{isempty}(\text{pul}) \)
                    if strcmpi(pul.getNodeName, 'amplitudes')
                        amp = pul.getFirstChild;
                        place = 1;
                        while \( \text{isempty}(\text{amp}) \)
                            if strcmpi(amp.getNodeName, 'amplitude')
                                if (place < 4)
                                    amp.setTextContent(num2str(newamp(place, place2)));
                                else
                                    amp.setTextContent('0');
                                end
                                place = place + 1;
                            end
                        end
                        amp = amp.getNextSibling;
                    end
                end
            end
        end
    end
end
place2 = place2 + 1;

pul = pul.getNextSibling;

pulse = pulse.getNextSibling;

node1 = node1.getNextSibling;
Appendix 2: Import

```csharp
using UnityEngine;
using UnityEditor;
using System;
using System.Linq;
using System.Collections.ObjectModel;
using PulseXmlReader;

public class Import : MonoBehaviour
{
    string blFilename;
    // String blFilename = "C:/Users/Gabriella/Documents/Plot12/
    // Assets/Resources/Bertil_4219.xml";
    String[] angles;
    public Collection<Pulse> pulses = new Collection<Pulse>();

    public Import(string name)
    {
        blFilename = name;
        openPulseDataFileAdapter();
    }

    private void openPulseDataFileAdapter()
    {
        Collection<String> tmpAngles = new Collection<String>();
        PulseData.openPulseDataFile(blFilename, ref pulses, ref tmpAngles);
        angles = tmpAngles.ToArray();
        // foreach (var item in pulses)
        // {
        //     Int32 skugga = item.peakVal[0];
        //     Int32 vinkel1 = item.peakVal[1];
        //     Int32 vinkel2 = item.peakVal[2];
        //     Int32 vinkel3 = item.peakVal[3];
        //     Console.WriteLine(skugga.ToString() + ", " +
        //                     vinkel1.ToString() + ", " + vinkel2.ToString() + ", "
        //                     + vinkel3.ToString() + ", ");
        // }
    }
}
```
39
40    }

Appendix 3: OpenFile

```csharp
using UnityEngine;
using UnityEditor;
using PulseXmlReader;

public class OpenFile : MonoBehaviour {
    public static string path;
    public Canvas CanvasD;
    public Canvas CanvasC;

    public void OpenExplorer() {
        path = EditorUtility.OpenFilePanel("", "", "xml");
        CanvasD.gameObject.SetActive(false);
        CanvasC.gameObject.SetActive(true);
    }
}
```
Appendix 4: CreateSphere

```csharp
using UnityEngine;
using PulseXmlReader;
using UnityEngine.Editor;
using System;
using System.Linq;
using System.Collections.ObjectModel;
using System.IO;
using UnityEngine.UI;

public class CreateSphere : MonoBehaviour {
    public GameObject cube;
    GameObject s;
    private Import import;
    public Collection<Pulse> pulses;
    public int nrOfPulses;
    float rotSpeed = 10;

    public Text Info;
    public Text Info2;
    public Text Info3;

    public Canvas CanvasA;
    public Button button1;
    public Button button2;
    public Button button3;
    public Button button4;

    public Canvas CanvasB;
    public Button baso;
    public Button debris;
    public Button eos;
    public Button gran;
    public Button lym;
    public Button Mono;
    public Button Rebuild;
    public Camera cam;

    public Canvas CanvasC;
    public Button orginal;
    public Button PCA;
```
private int one = 0;
private int two = 0;
private int three = 0;

public GameObject can1 = null;
public GameObject can2 = null;
public GameObject can3 = null;

private OpenFile file = null;
public string writeFile = null;

public string name = "";
private static System.Diagnostics.Process SDP;

private void Start()
{
    CanvasA.gameObject.SetActive(false);
    CanvasC.gameObject.SetActive(false);

    file = new OpenFile();

    CanvasB.gameObject.SetActive(false);
    // s = GameObject.CreatePrimitive(PrimitiveType.Cube);
    // s.transform.position = new Vector3(-3f, 2f, 7f);
    // s.transform.parent = cube.transform;

}

public void ChannelOneButton()
{
    if (one == 0)
    {
        one = 1;
    }
    else if (two == 0)
    {
        two = 1;
    }
    else
    {
        three = 1;
        CanvasA.gameObject.SetActive(false);
        Build();
    }
}
button1.interactable = false;

public void ChannelTwoButton()
{
    if (one == 0)
    {
        one = 2;
    }
    else if (two == 0)
    {
        two = 2;
    }
    else
    {
        three = 2;
        CanvasA.gameObject.SetActive(false);
        Build();
    }
    button2.interactable = false;
}

public void ChannelThreeButton()
{
    if (one == 0)
    {
        one = 3;
    }
    else if (two == 0)
    {
        two = 3;
    }
    else
    {
        three = 3;
        CanvasA.gameObject.SetActive(false);
        Build();
    }
    button3.interactable = false;
}

public void ChannelFourButton()
{
    if (one == 0)
    {
        one = 4;
    }
```csharp
else if (two == 0)
{
    two = 4;
}
else
{
    three = 4;
    CanvasA.gameObject.SetActive(false);
    Build();
}
button4.interactable = false;

public void BasoButton()
{
    GameObject[] cube1 = GameObject.FindGameObjectsWithTag("Baso");
    foreach (GameObject g in cube1)
    {
        GameObject.Destroy(g);
    }
}

public void DebrisButton()
{
    GameObject[] cube1 = GameObject.FindGameObjectsWithTag("Debris");
    foreach (GameObject g in cube1)
    {
        GameObject.Destroy(g);
    }
}

public void EosButton()
{
    GameObject[] cube1 = GameObject.FindGameObjectsWithTag("Eos");
    foreach (GameObject g in cube1)
    {
        GameObject.Destroy(g);
    }
}

public void GranButton()
```
GameObject[] cube1 = GameObject.FindGameObjectsWithTag("Gran");
foreach (GameObject g in cube1)
{
    GameObject.Destroy(g);
}

public void LymButton()
{
    GameObject[] cube1 = GameObject.FindGameObjectsWithTag("Lym");
    foreach (GameObject g in cube1)
    {
        GameObject.Destroy(g);
    }
}

public void MonoButton()
{
    GameObject[] cube1 = GameObject.FindGameObjectsWithTag("Mono");
    foreach (GameObject g in cube1)
    {
        GameObject.Destroy(g);
    }
}

public void RebuildButton()
{
    Destroy(cube);
    Build();
    cam.transform.position = new Vector3(0f, 0f, -1000f);
}

public void OriginalButton()
{
    name = OpenFile.path;
    CanvasC.gameObject.SetActive(false);
    CanvasA.gameObject.SetActive(true);
}

public void PCAButton()
{
    MonoScript ms = MonoScript.FromMonoBehaviour(this);
    string path = AssetDatabase.GetAssetPath(ms);
```csharp
FileInfo fl = new FileInfo(path);
string pathName = fl.Directory.ToString();
pathName.Replace('\', '/');
writeFile = pathName + "\pca.xml";
string basisFile = pathName + "\basis.txt";

SDP = new System.Diagnostics.Process();

string fileName = pathName + "\PCA.exe";
if (File.Exists(fileName))
{
    SDP.StartInfo.FileName = fileName;
    //SDP.StartInfo.WorkingDirectory = fileName;
    SDP.StartInfo.CreateNoWindow = true;
    SDP.StartInfo.UseShellExecute = false;
    SDP.StartInfo.RedirectStandardOutput = true;

    string send = "" + OpenFile.path + "" + " " + "" + writeFile + "" + " " + "" + basisFile + ""
    SDP.StartInfo.Arguments = send;
    SDP.Start();
    SDP.WaitForExit();

    StreamReader sr = new StreamReader(basisFile);
    string hela = sr.ReadLine();
    string[] words = hela.Split('?');

    string text1 = words[0];
    string text2 = words[1];
    string text3 = words[2];

    Info.text = text1;
    Info2.text = text2;
    Info3.text = text3;
}
CanvasC.gameObject.SetActive(false);
```
// Class1 pca = new Class1();

// MWArray s = pca.PCA(OpenFile.path, writeFile);

name = writeFile;

one = 1;
two = 2;
three = 3;

Build();

private void Build()
{
cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
cube.transform.position = new Vector3(0f, 0f, 0f);
cube.transform.localScale = new Vector3(700f, 700f, 700f);
cube.GetComponent<Renderer>().material = new Material(
    Shader.Find("Transparent/Diffuse"));
cube.GetComponent<Renderer>().material.color = new Color
    (0, 0, 0, 0.05f);

    // x-Axel
can1 = new GameObject();
can1.name = "TestCanvas1";
can1.AddComponent<Canvas>();
Canvas CanvasX = can1.GetComponent<Canvas>();
CanvasX.renderMode = RenderMode.WorldSpace;
CanvasX.gameObject.SetActive(true);
CanvasX.transform.parent = cube.transform;
can1.AddComponent<

Text xAxel = can1.GetComponent<Text>();
Font Arialfont = (Font)Resources.GetBuiltinResource(
    typeof(Font), "Arial.ttf");
xAxel.font = Arialfont;
xAxel.material = Arialfont.material;
xAxel.color = Color.black;
xAxel.fontSize = 20;

    // y-Axel
```csharp
    can2 = new GameObject();
    can2.name = "TestCanvas2";
    can2.AddComponent<Canvas>();
    Canvas CanvasY = can2.GetComponent<Canvas>();
    CanvasY.renderMode = RenderMode.WorldSpace;
    CanvasY.gameObject.SetActive(true);
    can2.transform.parent = cube.transform;
    can2.AddComponent<Text>();
    Text yAxel = can2.GetComponent<Text>();
    yAxel.font = ArialFont;
    yAxel.material = ArialFont.material;
    yAxel.color = Color.black;
    yAxel.fontSize = 20;

    // z-Axel
    can3 = new GameObject();
    can3.name = "TestCanvas2";
    can3.AddComponent<Canvas>();
    Canvas CanvasZ = can3.GetComponent<Canvas>();
    CanvasZ.renderMode = RenderMode.WorldSpace;
    CanvasZ.gameObject.SetActive(true);
    can3.transform.parent = cube.transform;
    can3.AddComponent<Text>();
    Text zAxel = can3.GetComponent<Text>();
    zAxel.font = ArialFont;
    zAxel.material = ArialFont.material;
    zAxel.color = Color.black;
    zAxel.fontSize = 20;

    CanvasB.gameObject.SetActive(true);

    switch (one)
    {
        case 1:
            xAxel.text = "Channel 1";
            break;
        case 2:
            xAxel.text = "Channel 2";
            break;
        case 3:
            xAxel.text = "Channel 3";
            break;
        case 4:
```
xAxel.text = "Channel 4";
break;
}
switch (two)
{
case 1:
    yAxel.text = "Channel 1";
    break;

case 2:
    yAxel.text = "Channel 2";
    break;

case 3:
    yAxel.text = "Channel 3";
    break;

case 4:
    yAxel.text = "Channel 4";
    break;
}
switch (three)
{
case 1:
    zAxel.text = "Channel 1";
    break;

case 2:
    zAxel.text = "Channel 2";
    break;

case 3:
    zAxel.text = "Channel 3";
    break;

case 4:
    zAxel.text = "Channel 4";
    break;
}
xAxel.transform.position = new Vector3(0f, -345f, -300f) ;

zAxel.transform.position = new Vector3(-345f, -345f, 0f) ;
Vector3 v = new Vector3(0f, 90f, 0f);
zAxel.transform.Rotate(v);
yAxel.transform.position = new Vector3(-300f, 0f, -300f) ;
Vector3 w = new Vector3(0f, 0f, 90f);
yAxel.transform.Rotate(w);

import = new Import(name);
pulses = import.pulses;
nrOfPulses = pulses.Count();

//Get max−min
float max0 = -100000000;
float max1 = -100000000;
float max2 = -100000000;
float max3 = -100000000;

float min0 = 100000000;
float min1 = 100000000;
float min2 = 100000000;
float min3 = 100000000;

foreach (var pulse in pulses)
{
    CellType cell = pulse.cellType;
    if (cell.Equals(CellType.Debris) == false)
    {
        if (pulse.peakVal[0] > max0)
        {
            max0 = pulse.peakVal[0];
        }
        if (pulse.peakVal[0] < min0)
        {
            min0 = pulse.peakVal[0];
        }
        if (pulse.peakVal[1] > max1)
        {
            max1 = pulse.peakVal[1];
        }
    }
if (pulse.peakVal[1] < min1)
{
    min1 = pulse.peakVal[1];
}

if (pulse.peakVal[2] > max2)
{
    max2 = pulse.peakVal[2];
}

if (pulse.peakVal[2] < min2)
{
    min2 = pulse.peakVal[2];
}

if (pulse.peakVal[3] > max3)
{
    max3 = pulse.peakVal[3];
}

if (pulse.peakVal[3] < min3)
{
    min3 = pulse.peakVal[3];
}

// get values for axis??
foreach (var item in pulses)
{
    float length = 0;
    float divide = 0;
    float minus = 0;

    float x = 0;
    float y = 0;
    float z = 0;
    // float x = pulses.First().peakVal[1];
    // float y = pulses.First().peakVal[2];
    // float z = pulses.First().peakVal[3];
switch (one)
{
    case 1:
        length = Math.Abs(max0 - min0);
        divide = length / 700;
        minus = (max0 - min0) / 2;
        x = item.peakVal[0];
        x = (x - min0 - minus) / divide;
        break;
    case 2:
        length = Math.Abs(max1 - min1);
        divide = length / 700;
        minus = (max1 - min1) / 2;
        x = item.peakVal[1];
        x = (x - min1 - minus) / divide;
        break;
    case 3:
        length = Math.Abs(max2 - min2);
        divide = length / 700;
        minus = (max2 - min2) / 2;
        x = item.peakVal[2];
        x = (x - min2 - minus) / divide;
        break;
    case 4:
        length = Math.Abs(max3 - min3);
        divide = length / 700;
        minus = (max3 - min3) / 2;
        x = item.peakVal[3];
        x = (x - min3 - minus) / divide;
        break;
}
switch (two)
{
    case 1:
        length = Math.Abs(max0 - min0);
        minus = (max0 - min0) / 2;
        divide = length / 700;
        y = item.peakVal[0];
        y = (y - min0 - minus) / divide;
        break;
    case 2:
        length = Math.Abs(max1 - min1);
        minus = (max1 - min1) / 2;

devide = lenght / 700;
y = item.peakVal[1];
y = (y - min1 - minus) / devide;
break;

case 3:
lenght = Math.Abs(max2 - min2);
minus = (max2 - min2) / 2;
devide = lenght / 700;
y = item.peakVal[2];
y = (y - min2 - minus) / devide;
break;

case 4:
lenght = Math.Abs(max3 - min3);
minus = (max3 - min3) / 2;
devide = lenght / 700;
y = item.peakVal[3];
y = (y - min3 - minus) / devide;
break;

switch (three)
{
case 1:
lenght = Math.Abs(max0 - min0);
minus = (max0 - min0) / 2;
devide = lenght / 700;
z = item.peakVal[0];
z = (z - min0 - minus) / devide;
break;

case 2:
lenght = Math.Abs(max1 - min1);
minus = (max1 - min1) / 2;
devide = lenght / 700;
z = item.peakVal[1];
z = (z - min1 - minus) / devide;
break;

case 3:
lenght = Math.Abs(max2 - min2);
minus = (max2 - min2) / 2;
devide = lenght / 700;
z = item.peakVal[2];
z = (z - min2 - minus) / devide;
break;

case 4:
lenght = Math.Abs(max3 - min3);
```csharp
minus = (max3 - min3) / 2;
devide = length / 700;
z = item.peakVal[3];
z = (z - min3 - minus) / devide;
break;
}

// foreach (var item in pulses)
//{
//    float x = item.peakVal[one - 1];
//    float y = item.peakVal[two - 1];
//    float z = item.peakVal[three - 1];
//    // float x = pulses.First().peakVal[1];
//    // float y = pulses.First().peakVal[2];
//    // float z = pulses.First().peakVal[3];

//    switch (one) {
//        case 1:
//            x = (x - 2500) / 8;
//            break;
//        case 2:
//            x = (x - 2300) / 7;
//            break;
//        case 3:
//            x = (x - 2300) / 7;
//            break;
//        case 4:
//            x = (x - 50) * 7;
//            break;
//    }
//    switch (two)
//    {
//        case 1:
//            y = (y - 2500) / 8;
//            break;
//        case 2:
//            y = (y - 2300) / 7;
//            break;
//        case 3:
//            y = (y - 2300) / 7;
//            break;
//        case 4:
//            y = (y - 50) * 7;
//            break;
//    }
```
switch (three) {
    case 1:
        z = (z − 2500) / 8;
        break;
    case 2:
        z = (z − 2300) / 7;
        break;
    case 3:
        z = (z − 2300) / 7;
        break;
    case 4:
        z = (z − 50) * 7;
        break;
}

GameObject g = GameObject.CreatePrimitive(PrimitiveType.Sphere);
g.transform.position = new Vector3(x, y, z);
g.transform.parent = cube.transform;
g.transform.localScale = new Vector3(0.01f, 0.01f, 0.01f);

CellType cell = item.cellType;

if (cell.Equals(CellType.Baso)) {
    // g.GetComponent<Renderer>().material = new Material(Shader.Find("Transparent/Diffuse"));
    g.GetComponent<Renderer>().material.color = new Color(0, 0, 1, 1f);
    g.tag = "Baso";
}
else if (cell.Equals(CellType.Debris)) {
    // g.GetComponent<Renderer>().material = new Material(Shader.Find("Transparent/Diffuse"));
    g.GetComponent<Renderer>().material.color = new Color(0, 1, 1, 1f);
    g.tag = "Debris";
}
else if (cell.Equals(CellType.Eos))
{
    //g.GetComponent<Renderer>().material = new
    Material(Shader.Find("Transparent/Diffuse"));
    g.GetComponent<Renderer>().material.color = new
    Color(0, 1, 0, 1f);
    g.tag = "Eos";
}
else if (cell.Equals(CellType.Gran))
{
    //g.GetComponent<Renderer>().material = new
    Material(Shader.Find("Transparent/Diffuse"));
    g.GetComponent<Renderer>().material.color = new
    Color(1, 0, 1, 1f);
    g.tag = "Gran";
}
else if (cell.Equals(CellType.Lym))
{
    //g.GetComponent<Renderer>().material = new
    Material(Shader.Find("Transparent/Diffuse"));
    g.GetComponent<Renderer>().material.color = new
    Color(1, 0, 0, 1f);
    g.tag = "Lym";
}
else if (cell.Equals(CellType.Mono))
{
    //g.GetComponent<Renderer>().material = new
    Material(Shader.Find("Transparent/Diffuse"));
    g.GetComponent<Renderer>().material.color = new
    Color(1, 1, 1, 1f);
    g.tag = "Mono";
}
else if (cell.Equals(CellType.Undef))
{
    GameObject.Destroy(g);
}

//Mesh mesh = g.GetComponent<MeshFilter>().mesh;
//mesh.RecalculateBounds();
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712

// x = (Int32) (x - 1000) / 3;
// y = (Int32) (y - 2611) / 8;
// z = (Int32) (z - 2709) / 8;

// GameObject g = GameObject.CreatePrimitive(
//   PrimitiveType.Sphere);
// g.transform.position = new Vector3(x, y, z);
// g.transform.parent = cube.transform;
// count += 1;
//

// s = GameObject.CreatePrimitive(PrimitiveType.Sphere);
// s.transform.position = new Vector3(40, 40, 40);
// s.transform.localScale = new Vector3(10, 10, 10);

private void Update()
{
  if (Input.GetMouseButton(0))
  {
    float rotX = Input.GetAxis("Mouse X") *
               rotSpeed * Mathf.Deg2Rad;
    float rotY = Input.GetAxis("Mouse Y") *
               rotSpeed * Mathf.Deg2Rad;

    cube.transform.RotateAround(Vector3.up, -rotX);
    cube.transform.RotateAround(Vector3.right, rotY);
  }
}
//public void SliderX()
//{
//  if (sliderX.value == 0)
//  {
//   }
//   float xMove = sliderX.value - oldValue;
//   GameObject[] cube1 = GameObject.FindGameObjectsWithTag
//                ("Cell");
//   foreach (GameObject g in cube1)
//   {
//     if (g.tag != "Text")
//     {
//       float newPosition = g.transform.position.x + (g.transform.position.x * xMove);
//       Vector3 vec = new Vector3(newPosition, g.
//                      transform.position.y, g.transform.position.z);
//       float step = speed * Time.deltaTime;
//       g.transform.position = Vector3.MoveTowards(vec
//                      , g.transform.position, step);
//     }
//     }
//     oldValue = sliderX.value;
//  }

// Update is called once per frame
}
using UnityEngine;
using UnityEditor;

public class Zoom : MonoBehaviour
{

    //public float zoom = 1F;
    Vector3 newPosition;

    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {

        Camera cam = GetComponent<Camera>();
        RaycastHit hit;
        Ray ray = cam.ScreenPointToRay(Input.mousePosition);
        Vector3 targetPos = new Vector3(0f, 0f, 0f);

        if (Physics.Raycast(ray, out hit))
        {
            if (Input.GetMouseButtonUp(1))
            {
                targetPos = hit.point;
                // zoom = 1000f;
                // cam.orthographicSize = zoom;
                cam.transform.position -= (targetPos * 10) / 30f;
            }
            if (Input.GetMouseButtonUp(2))
            {
                cam.transform.position = new Vector3(0f, 0f, -1000f);
            }
        }
    }
} // transform.position }
}