| S | 17 |
|---|---|


**De La Salle University • College of Computer Studies**

Advanced Discrete Structures (ADVDISC)

AY 2015-2016 Term 1

# Use of Matrices in Computer Graphics

**Submitted by:**

Aquino, Kurt Neil

Esquillo, Lance Patrick

Gomez, Miguel Francisco

**Submitted on:**

October 26, 2015

**Introduction**

Matrices provide a solid, general tool to represent and combine all common transformations. Basic transformation operations such as translation, rotation, scale, etc. can be accomplished without the use of matrices as well; although in order to provide the general representations for types of transformation, matrices become a very useful choice. The biggest advantage of using matrices is the capability to combine and concatenate transformations; this is common operation in computer graphics, so it is beneficial to have a single representation and a single function to combine all sorts of transformations.

The use of matrices in computer graphics has been applied for more than 3 decades. Many industries involving architecture, animation, automotive, etc. that were formerly done by hand drawing now are done routinely with the aid of computer graphics. The earliest to heavily rely on computer graphics is the video gaming industry; which by now are able to represent rendered polygons in 3-Dimensions. In the video gaming industry, matrices are major mathematic tools to construct and manipulate a realistic animation of a polygonal figure.

For this machine project, out task is to provide an application which lets the user to plot 2 dimensional objects such as points, line segments, vectors, polygons, and conics and allow them to perform basic transformation functions such as translate, rotate, reflect, etc.

**Implementation**

For this machine project, our group decided to use Matrix Laboratory, or better known as MatLab, as the environment. MatLab is a proprietary programming language developed by MathWorks, it allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, Fortran and Python.

The reason why our group decided on using MatLab is due to its free form structure and syntax similar to that of Python's. Another reason is that MatLab is considered as a dedicated programming environment for dealing numerical computations. Along with the use of MuPad and Simulink, which are additional packages included in MatLab, matrix representations as well as multi-domain simulations can be done easily.

For the user interface, our group made use of the GUI editor, named GUIDE, provided by MatLab. Although MatLab already has a library for majority of the matrix transformation functions, our group did not use any of them as it is against the MP specifications. The only methods our group used was the basic math operations (sin, cos, sqrt, etc.), plotting methods and its other variants, along with some GUI functionalities for the screen flow. The rest of the methods and functions for the matrix representations and operations included in the code were programmed by ourselves.

**Design**

Our code is composed of two modules, with each module handling its own set of functions. The first is the **Matrix Representation Module**. This module deals with the gathering of the parameters required for the specified object the user would like to plot (e.g. Object: Parabola, Inputs: x Vertex, y Vertex, magnitude, orientation) in terms of the object's equation. Depending on the type of object the user has selected, it then loads the transformation operations available for that object. (e.g. Object: Point, Transformations: Translate only).

Variables Used:

- **objectType** – String used to determine which type of object was selected.
- **functionType** – String used to determine which type of transformation function was selected.

Some of the methods used in this module are:

- **setObject(objectType)** – determines the type of object the user has selected from the combo box. It then loads the panel containing the parameters required to be filled with respect to the selected object.
- **setObjectFunctions(objectType)** – loads the available transformation functions for the selected object.
- **plotObject(objectType)** – plots the representation of the selected object equivalent to its equation to the main axes. (e.g. Object: Hyperbola, Equation: ((x-h)/a)^2 – ((y-k)/b)^2 = 1, Plot Representation: x = -100:1:100, y = sqrt(b^2 * (x-h).^2 / a^2 - b^2 ) + k)
- **restoreOriginal(objectType)** – clears the main axes and plots the original object.

The second module is the **Matrix Manipulation Module**. This modules deals with the transformation functions available for the selected object. As mentioned before, each object has its own set of transformation functions, with each function having its own set of parameters depending on the operation (e.g. Operation: Translate, Parameters: x Coordinate, y Coordinate). It the loads the transformed object into the main axes, along with the original object.

Variables used:

- **objectType** – String used to determine which type of object was selected.
- **functionType** – String used to determine which type of function was selected.
- **func** – Integer used to determine which type of operation is to be used for the function.
- **objectVal** – Array of values containing the parameters of the original object.
- **transformVal** – Array of values containing the parameters of the transformation function.
- **plotData** – Plot Representation of the newly transformed object.

Some of the methods used in this module are:

- **setFunction(functionType)** – determines the type of operation the user would like to apply on the created object. It then loads the panel containing the parameters required to be filled with respect to the selected operation.
- **func = getTransform(functionType)** – gets the data from parameters of the selected transformation function and returns an identifier for the type of operation to be performed.
- **transformObject(objectType, functionType, func)** – applies the selected transformation function on the created object then plots it into the main axes. This function calls other methods specific to the type of operation to be applied on the specified object.

- **plotData = <functionType><objectType>(objectVal, transformVal, func)**
  - Returns the plot representation of the newly transformed object after performing specific operations on the original object with respect the type of transformation
  - ex: plotData = translatePoint(objectVal, transformVal, func)

```
if (func == 1)                              %confirming function identifier
    x = objectVal{1} + transformVal{1};     %translating x Coordinate
    y = objectVal{2} + transformVal{2};     %translating y Coordnate
    plotData = plot(x, y);                  %returns transformed plot data
end
```

**Conclusion**

With this machine project, our group was able to present one of the many applications of matrices in computer graphics. By representing matrices as digital and visual figures, our group was able to manipulate specified points in order to create new figures with respect to the original. In our group's implementation, rather than dealing with matrices only, we also dealt with the manipulation of objects by performing operations on the coefficients of the parameters of their equation forms. Matrix manipulation can also be applied on 3D figures, although it will involve even more complex operations for the transformation functions.

**Contribution of Members**

Aquino, Kurt Neil

- Background Research on MatLab and Transformation Functions
- GUI Design and Functionality
- GUI Testing and Debugging
- Plotting of Conics
- Transformation Functions for Conics
- Introduction and Design Section of the Paper

Esquillo, Lance Patrick

- Background Research on MatLab and Transformation Functions
- GUI Testing and Debugging
- Plotting of Line Segments and Polygons
- Transformation Functions for Line Segments, Polygons, and Conics
- Implementation and Conclusion Section of the Paper

Gomez, Miguel Francisco

- Background Research on MatLab
- GUI Testing and Debugging
- Plotting of Points and Vectors
- Transformation Functions for Points and Vectors

**References**

MATLAB. (n.d.). Retrieved October 25, 2015, from https://en.wikipedia.org/wiki/MATLAB.

Yip, T. (2001, December 3). Matrices in Computer Graphics. Retrieved October 25, 2015, from http://www.math.washington.edu/~king/coursedir/m308a01/Projects/m308a01-pdf/yip.pdf.

DeHaan, M. (2013, December 17). Matrix Mathematics: How Do We Use Matrices In Day-to-Day Life? Retrieved October 25, 2015, from http://www.decodedscience.org/practical-uses-matrix-mathematics/40494.

Transformation Matrix. (2011, July 19). Retrieved October 25, 2015, from http://mathforum.org/mathimages/index.php/Transformation_Matrix.