XL CILAMCE
IBERO-LATIN AMERICAN CONGRESS ON COMPUTATIONAL METHODS IN ENGINEERING
NOVEMBER 11-14, 2019
Praiamar Natal Hotel & Convention Natal, RN-BRAZIL

MOBILE APPLICATION DEVELOPMENT FOR LEARNING MECHANICS OF MATERIALS

Pedro P. Sarkis Rosa
Ramon Pereira da Silva
pedrosarkis.1@gmail.com
ramon@ufmg.br
Federal University of Minas Gerais
Av. Antônio Carlos 6627, Pampulha, 31270-901, Belo Horizonte, Minas Gerais, Brazil

Abstract. This work presents *Mechanics of Materials 3D*, a mobile application initially targeting the Android platform. This app is the result of a beginner's research program at the Federal University of Minas Gerais, whose objective is to design a tool to help students to learn elementary concepts of Mechanics of Materials, especially in the study of stress states. The fundamental idea of the project was to make knowledge more accessible to students by allowing abstract concepts to be better understood and absorbed through playful interaction and graphic representation. It is an attempt to reconcile traditional teaching with new learning possibilities provided by technology, with a special focus on the new generations of students enrolling at universities every year. The application uses the matrix library JAMA to perform the diagonalization of the stress tensor provided, thereby obtaining the principal stresses and corresponding directions. In sequence, it gives access to graphical interfaces that present in an intuitive and interactive way these results, as well as other results such as the Mohr's circle and the infinitesimal cube in different configurations (state supplied by the user, principal stresses and maximum shear stress). These interfaces were elaborated through the synthesis of Java programming, layout design in the XML markup language, the Android Canvas class along with the OpenGL ES graphic API.

Keywords: Mobile application, Mechanics of Materials, Learning tool, Android, Stress states

1 Introduction

Mechanics of Materials 3D is an application developed for mobile devices, initially targeting the Android platform. It aims to be a tool that helps in learning elementary concepts of Mechanics of Materials, especially regarding the study of stress states. The application is the result of a volunteer beginner's research program at the Federal University of Minas Gerais (UFMG) and it is based on the attempt to reconcile traditional teaching with new learning possibilities provided by technology, with a special focus on the new generations of students enrolling universities every year, who are increasingly demanding this kind of initiative. Despite its recent release, the *Mechanics of Materials 3D* application is already relatively successful and has been downloaded by people from over 40 different countries from all continents.

In the application, after inputting the stress tensor, the user receives results such as the principal stresses and their respective directions, the maximum shear stress, as well as access to graphical interfaces that present these and other results in an intuitive and playful way: the 3D representation of Cauchy's infinitesimal cube of the given stress state, which can assume different stress configurations; the Mohr's circle; an interface joining both in the same environment to see how they relate. Strictly speaking, the tool can be extended to other cases involving tensor quantities, such as deformations and moments of inertia, being second-order symmetric tensors too (Boresi and Schmidt [1]).

2 Interfaces

The application is divided into different interfaces, each contributing differently to the learning of mechanics of materials. Following is a brief explanation of each and the main aspects of its implementation.

2.1 Interface 1: tensor input

In the study of stresses acting on arbitrary planes passing through an internal point *O* of a body subject to external forces, one finds three mutually perpendicular planes in which the shear stress disappears, leaving only normal stress components, called principal stresses. In solving the problem of finding these stresses and their directions, one falls into equations equivalent to the standard eigenvalue problem, in which the principal stresses are the eigenvalues and their directions the respective eigenvectors (Boresi and Schmidt [1]).

When the user opens the application, he is asked to provide a stress tensor as input. Only the six independent stress components are required from the user because listeners (`TextWatcher`) have been implemented to fill the remaining components. From this, the principal stresses and directions are returned as output, as well as the maximum shear stress and its directions. For code efficiency reasons, it was decided to implement the solution of this problem through the eigenvalues and eigenvectors approach. For that, the library JAMA (A Java Matrix Package) was implemented [2].

Then, in the bottom corner of the interface, the user is provided with a menu to choose from the following three interfaces.

2.2 Interface 2: infinitesimal cube

In this interface, the user has access to an interactive graphical representation of the infinitesimal stress state cube, where the user can control the camera via touch screen and observe the cube from any desired angle. Initially, it is representing the tensions supplied by the user, inserted in the Cartesian coordinate space and with the respective scaled vectors. The user can access via buttons the representation of two other states: principal stresses and maximum shear stress, causing the cube to rotate in the coordinate system and the stress vectors to assume different configurations. The application aims to be the first in the Play Store to feature the infinitesimal cube 3D rendering with this range of options and user interactions.

The cube's 3D rendering was done via OpenGL (Open Graphics Library), which is a widely used free graphical API (Application Programming Interface). The subset of OpenGL used in mobile applications is OpenGL ES (for Embedded Systems). For this application, we used the OpenGL ES 1, which is the oldest and simplest version. This version was chosen because rendering the cube environment is not complex, eliminating the need for shaders - which are computer programs written in OpenGL Shading Language (GLSL) and which are mandatory for the latest versions of the API. Another reason is that this version has the possibility of working with matrix stacks, functionality that is deprecated in the most current versions, but whose functionality has been useful for our purposes, as will be seen.

This 3D rendering is made up of the synthesis of the following independent objects: a cube, square prisms (base of the vector), square pyramids (top of the vector), lines (axes), and the letters *x, y,* and *z* (legend of the axes). Each of these objects is described in a different Java class that contains its vertex and color arrays. These vertices are arranged so that they form triangles since all objects in OpenGL are composed of the junction of triangles. The more detailed the geometry of the object, the more triangles are needed to describe it. Once you have the necessary spatial objects, just insert them into a common space with the other objects and position them properly, which is done through matrix transformations acting on the vertex matrix.

One of the core mechanics of 3D engines is the chain of matrix transformations that allows to represent a 3D object on a 2D monitor (M. Alamia [3]). They are necessary for remapping between different vector spaces, progressing from the independent conception of the objects to their joint projection on the device screen. Objects are initially designed in an independent vector space called Model Space and are then inserted via matrix transformations into a common space called World Space. Afterward, the observer (the camera) is inserted into the 3D space through transformations that make up the View Space. Finally, there is the transformation that projects the 3D onto the 2D display, leading to the Projection Space. Because each object has a distinct positioning in common space, each object has its own matrix transformations, which can be concatenated by dot products to give rise to a single transformation matrix.

All of these transformations in the application are inserted into a matrix stack, which is a stack of transformation matrices. The main reason to use them is that they “make it easy to create and manage complex hierarchical objects and scenes, where transforms can be built upon (and removed from) other transforms” (Gordon and Clevenger [4]). For example, one can request an object to be drawn when the matrix stack is in a configuration "A", then make changes to add and remove transformations from the stack, commanding after another object is drawn in a configuration "B". This makes it easy to control which transformations will be performed on which objects by placing those transformations common to all at the bottom of the stack – easily synchronizing their movements – and the most specific to each object at the top of it. At the base of the matrix stack of this rendering are the transformations that govern camera movement, since all objects are subjected to these transformations and move together when the user changes the view. Next, the first objects to be drawn are the axes. In sequence, the letters *x*, *y*, *z* are drawn, adding to them rotation transformations opposite to the camera rotation, so that they always remain facing the viewer. Then the rotation matrix that gives the orientation of the cube is added to the stack, composed by the directions of the stresses calculated in the initial interface, so the cube can be drawn with the proper rotation. This transformation is kept in the stack so that it remains valid for the next objects to place: the vectors. They consist of two objects, the prismatic base of the vector and its pyramidal head. For each vector the base is positioned with the proper orientation on the side of the cube and, in sequence, a scaling transformation is applied in the direction of the base length according to the magnitude of the vector. This dimension factor is calculated as follows: (*i*) the largest allowable value is the cube edge measurement subtracted from the vector head measurement so that the shear vectors never overlap; (*ii*) this vector dimension is attributed to the principal stress with the largest modulus, conditioning the size of the others proportionally. Finally, the vector head is positioned, repeating the process from the beginning for each one.

2.3 Interface 3: Mohr's circles

From the principal stresses a diagram can be constructed that facilitates a graphical visualization of the stress states and its transformations: the Mohr's circle. In the case of three dimensions, there are three circumferences (which may overlap), in which the intersections of the circumferences with the abscissa axis are the principal stresses, and the maximum shear stress corresponds to the radius of the largest circumference.

In this application interface, the principal stresses information is used to draw Mohr circles for the 3D case. We used a native Android class to do the drawing, the `Canvas`. By determining different colors, styles, transparencies and stroke widths, we created the representation of the circles, whose interactivity is through buttons on the layout that allow the user to select the circle they want to visualize and get their coordinate information, clearly and pleasantly.

For aesthetic reasons, it was determined that the radius of the largest circumference is fixed, being equal to the height of the device screen divided by three, and the other radii are calculated proportionally. As for the positioning of the elements, we have that the abscissa axis is fixed in the middle of the screen, while the circumferences and the ordinate axis are positioned according to three distinct scenarios: all positive principal stresses, all negative, mixed. For each of these cases, the positioning of the elements is fixed, regardless of the magnitude of the stresses. This is because it was preferred to value the aesthetics and simplicity of the results, rather than the correct scaling of the circumferences on the abscissa axis, which could lead to their great distancing from the origin in the case of high moduli that are close to each other (leading to large horizontal displacement and small circumference radius).

2.4 Interface 4:

It is the junction of both previous interfaces in one environment so that the user can observe how they relate to each other in different stress configurations. For this, the points relative to the stresses represented in the infinitesimal cube are plotted in the Mohr's circles, using colors compatible with those used on the cube's faces for easier comprehension.

3 Android implementation[1]

Android is a Linux-based operating system for mobile devices developed by Google. Originally, its application programming was done predominantly in Java, but currently different languages are also used, such as Kotlin. For *Mechanics of Materials 3D*, however, we opted for programming entirely in Java.

An Android application is built through app `Components`. There are different types of components, but our application uses only `Activities`, which are each of the different application screens. For example, in this application the file `TensorActivity.java` has the source code that governs the behavior of the application's first screen, implementing the functions that perform the stress calculations, as well as dialoguing with the XML file containing the UI (User Interface) layout. An activity is implemented as a subclass of the Android native `Activity` Java class. In this application, the activities communicate with each other through setters and getters, according to the logic of object-oriented programming. This application has six activities because in addition to the interfaces presented there are an About page and another one for credits.

In addition to Java code, `Resources` separate from source code are also used. Among them, we have `Layouts`, whose design is made via XML markup language. Each of the activities, when initialized, in its `onCreate()` method can pull its specific layout file, and the dialogue between Java code and layout elements is done by assigning IDs to the elements. In addition to layout features, the application uses other resources such as `Drawables` containing the icon images used in the buttons.

The basic building block for UI elements of layouts is the `Views`, which is the base class for

[1] All elements of the Android environment listed below are in its official documentation [5].

widgets, that are used to create buttons, text fields, etc. Text displayed on the screen, for example, is made via `TextView`, whereas text input boxes are made via `EditText` and displayed images are via `ImageView`. In addition, you can determine numerous different attributes of each view, and their placement in the layout has been done through `Constraint Layout`, which allows you to position elements entirely by drag-and-dropping by using the Android Studio IDE (Integrated Development Environment), instead of editing their position in the XML. In addition to the standard views, some custom views have been implemented in the application, the two main ones being a customization of the `GLSurfaceView` (which contains the OpenGL rendering display) so that we could work with receiving information via touch screen; and a custom view that extends directly from the `View` native class adapted to receive `Canvas` drawings as needed for our purposes. A final touch on the UI was the implementation of a Floating Action Button to receive the menu that leads to other interfaces, made available in open source by a collaborator via public sharing on GitHub [6].

4 Future of the project

The project will continue with the production of new applications, such as the calculation of geometric properties of flat areas and the representation of internal loadings in framed structures as a game. In addition, it is intended to continue improving *Mechanics of Materials 3D* by implementing several upgrades, such as the inclusion of classic failure criteria, the strain states and their corresponding 3D rendering and the possibility that the cube can be rotated in any desired orientation (without rotating the coordinate system) so the change in the representation of the stress state vectors can be observed to any desired stress state in real-time.

Acknowledgements

We thank the UFMG volunteer beginner's research program for the opportunity to participate in this project. We also thank André Sarkis Rosa for being the full-time Beta user and for the countless suggestions. Lastly, thanks to the entire online community that shares knowledge publicly for free. If it were not for these contributors, active on forums like Stack Overflow and dozens of blogs, this app would not exist today.

References

[1] A. P. Boresi and R. J. Schmidt. *Advanced Mechanics of Materials*. John Wiley & Sons, 2003.
[2] J. Hicklin et al. (2012). *JAMA: Java Matrix Package*. Available at: https://math.nist.gov/javanumerics/jama/ [Accessed 7 Aug. 2019].
[3] M. Alamia. *Article - World, View and Projection Transformation Matrices* [Blog post]. Achieved version available at http://web.archive.org/web/20190330053238/http://www.codinglabs.net/article_world_view_projection_matrix.aspx [Accessed 7 Aug. 2019].
[4] V. S. Gordon and J. Clevenger. *Computer Graphics Programming in OpenGL with Java*. Mercury Learning and Information, 2017.
[5] Android Developers. (2019). Documentation. Available at: https://developer.android.com/docs [Accessed 7 Aug. 2019].
[6] D. Tarianyk, FloatingActionButton, (2015), GitHub repository, https://github.com/Clans/FloatingActionButton