XLVI CILAMCE
VITÓRIA - ES - BRAZIL - 2025
XLVI IBERO-LATIN AMERICAN CONGRESS ON COMPUTIONAL METHODS IN ENGENEERING

Experiences on Teaching Computer Graphics for Engineers and Scientists

Andre M. B. Pereira[1], Luiz Fernando Martha[2]

[1]*Instituto de Computação, Universidade Federal Fluminense*
Rua Passo da Patria 156, 24210-240, Niteroi/RJ, Brazil
andre@ic.uff.br
[2]*Dept. of Civil Engineering, Pontifícia Universidade Católica do Rio de Janeiro*
Rua Marquês de São Vicente 225, 22451-900, Rio de Janeiro/RJ, Brazil
lfm@tecgraf.puc-rio.br

Abstract. In this paper, we describe practical experiences, acquired over several years, in the development of a graduate level course in applied computer graphics. Since the target audience of the course is engineers and scientists, we focus mainly on topics related to pre-processing and post-processing techniques, which, although fundamental for performing numerical simulations, are usually neglected in the educational process. The course structure covers basic topics, such as: representation of curves and surfaces; brief introduction to computational geometry; mesh generation; modeling and representations of solids; and overview of visualization techniques. Our main challenge is to teach very complex subjects that students will be exposed to for the first time in this course. However, students who pursue such courses are usually at a high level of maturity and understand that the topics covered are fundamental for their career and research. Therefore, to give an effective and practical character to the course, we follow an approach that is completely based on the development of practical software with user interfaces, relying on open source packages with Python APIs. We chose to use Qt and OpenGL to create interactive graphical user interfaces in a hands-on scheme. The course starts with the development of a curve collector and ends with a full-functional 2D mesh generator. Experiences acquired over the years are combined to achieve the best performance and learning process. A taste of the course can be found at https://web.tecgraf.puc-rio.br/~lfm/compgraf-251/.

Keywords: educational, geometric modeling, computer graphics, teaching, mesh generation.

1 Introduction

Computer simulation in science and engineering are indispensable tools in the design process of projects in order to optimize time and other required resources. One of the fundamental steps of a computational simulation is the generation and visualization of a model that represents the problem to be simulated and this naturally includes computer graphics. This model consists of a geometric representation (e.g., points, curves and surfaces), attributes (e.g. boundary conditions and materials) and meshes (e.g. set of triangles and/or quadrilaterals and nodes), which can be generated by several means, including manually. However, in most engineering problems, especially structural and geotechnical, the geometry is not simple, and may contain constraints, inclusions, holes and cracks that should be taken into account in the modeling process. Therefore, an automatic mesh generation is needed which is able to represent the geometry of the problem accurately, especially when the shape is complex.

The proposed course aims to introduce fundamentals of computer graphics in simulations in order to deal with the problems described above. The course structure involves and covers the following topics: representation of curves and surfaces; brief introduction to computational geometry; topological data structures; mesh generation: structured and unstructured; Delaunay triangulation and boundary propagation; modeling and representations of solids: models by space decomposition and B-Rep; overview of visualization techniques.

The course starts with the development of a curve collector and ends with a full-functional 2D mesh generator. Experiences acquired over the years are combined to achieve the best performance and learning process. A version of the course can be found at https://web.tecgraf.puc-rio.br/~lfm/compgraf-251/.

2 Course on Computer Graphics

The course was designed to last 12 weeks, which includes 24 lectures of up to 2 hours each or in a condensed format of 12 lectures of 3 hours each, as it is typically taught at PUC-Rio. In the next subsections, we detail the topics approached in every lecture, based on the course format of 24 lectures.

2.1 Presentation of the lectures

In the first lecture, we introduce the full course to the students, discussing the topics that will be approached and the rules on how they will be assessed. We state that the main aim of the course is the introduction to computer graphics concepts and algorithms [1] for students who intend to develop and customize technical-scientific software.

2.2 Presentation of the development environment

The programming language used in the course is Python, currently in version 3.13.6. The program compilation and creation environment (IDE) used in this course is Visual Studio Code. In addition to Python and Visual Studio Code, the course requires the installation of the Qt graphical user interface system, version 6, the OpenGL graphics system, and some other additional libraries, which can be installed as indicated in the lecture notes. While PySide6 is the Qt official binding, PyQt (specifically PyQt6) is another popular and well-established third-party set of Python bindings for Qt. Both PySide and PyQt offer similar functionalities. To begin to acquire familiarity with the development environment, we programmed a calculator which adds two numbers, together with the students, as illustrated in Fig. 1.

```
from PyQt5.QtWidgets import QMainWindow
from gui.myapp import Ui_MyApp

class AppController(QMainWindow, Ui_MyApp):
    def __init__(self):
        super().__init__()
        super().setupUi(self)

        # Connecting the signal
        self.calculateButton.clicked.connect(self.calculate)

    # @Slot()
    def calculate(self):
        a = float(self.firstNumber.text())
        b = float(self.secondNumber.text())
        self.resultNumber.setText(str(a+b))
```

```
import sys
from PyQt5.QtWidgets import QApplication
from appcontroller import AppController

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ctrl = AppController()
    ctrl.show()
    app.exec_()
```

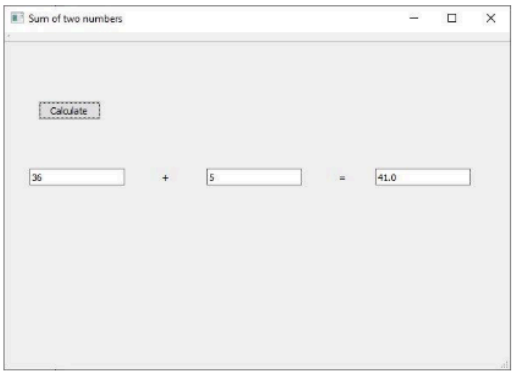


Figure 1. Program to add two numbers with GUI.

2.3 Introduction to computer graphics for scientists and engineers

In this lecture, we establish our goals and make a plan to achieve them. This means that we need to design an event driven graphical environment, step-by-step, to be able to iteratively build our geometric models. Figure 2 illustrates the kind of scientific tool we desire to build with knowledge taught in this course.

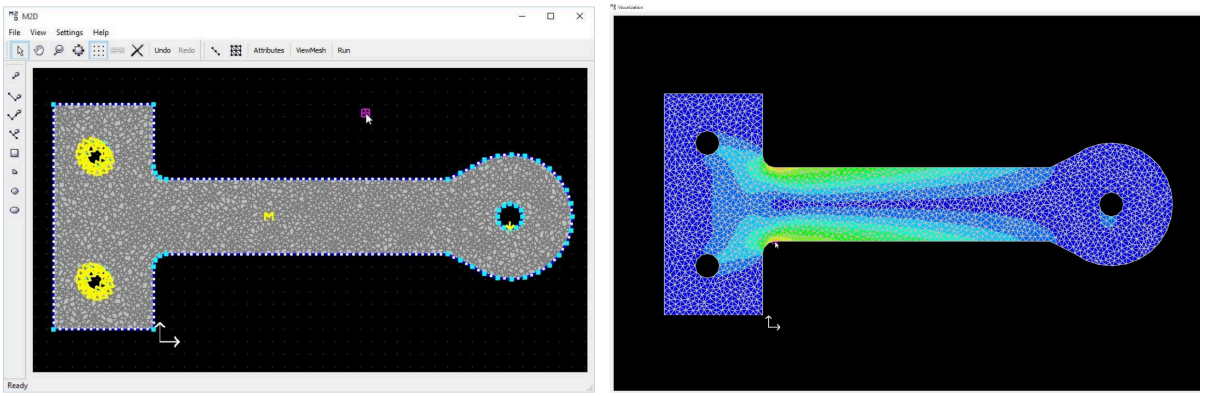


Figure 2. Scientific tools to perform computational simulations.

2.4 Programming in an event driven iterative environment

In this lecture, the students begin to dive deep into getting their hands-on GUI applications. It starts with the development of a calculator with RPN in Python. Its first version focuses on a console application. This is a very challenging problem, as it involves many iterative events and the need for specific data structures [2] to store and access the generated data. Figure 3 illustrates an example of RPN calculator and its console version.

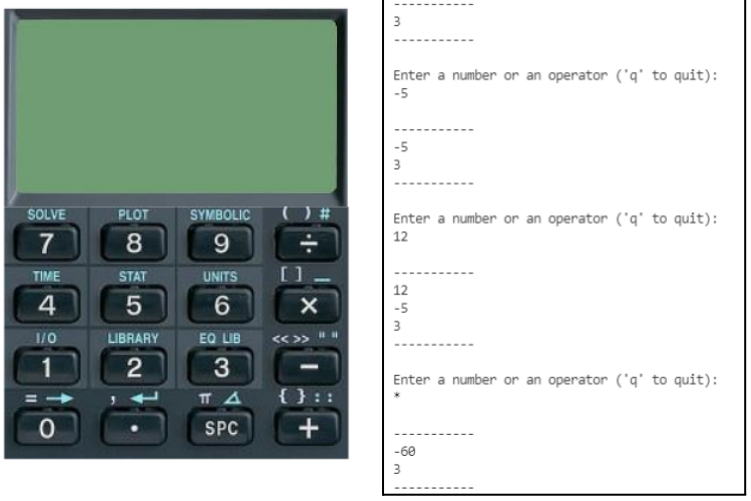


Figure 3. Idealization of a RPN calculator (left) and its console version built in Python (right).

2.5 Review of object oriented modeling

The RPN calculator is an excellent example to demonstrate the importance of object oriented modeling for designing powerful and robust applications. Object oriented modeling allows us to move smoothly to a fully functional RPN application with an interactive graphical user interface consisting of several widgets, such as push buttons. Here, we have the opportunity to review the basic concepts of object oriented programming, as well as UML diagrams and design patterns. It is also a good moment to propose the first Homework.

2.6 Two-dimensional Visualization with OpenGL

In this lecture, we introduce the fundamentals of the traditional standard graphic library OpenGL. We use as a hands-on activity the development of a program to render a triangle on the screen, as illustrated in Fig. 4.

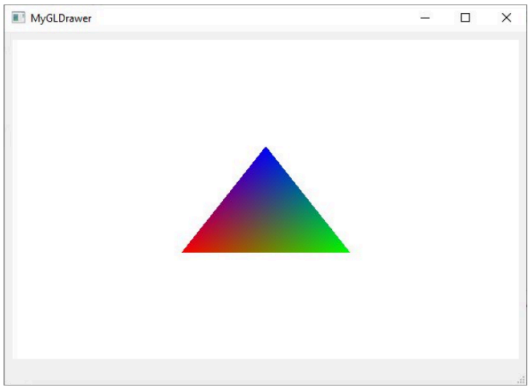


Figure 4. Triangle drawer with OpenGL.

2.7 Geometric Transformation in 2D and Window-Viewport

Geometrical transformations (such as translations, rotations and scaling) are fundamentals to positioning geometric objects on the screen, but also to carry out the basic setup of the graphical environment. In this lecture, we present the basic definitions of geometric transformations [3] (Fig. 5) and we focus on the world coordinate system and screen coordinate system; world window, interface window, and viewport; window-to-viewport mapping; window-to-viewport transformation in OpenGL; and 2D rendering pipeline in OpenGL.

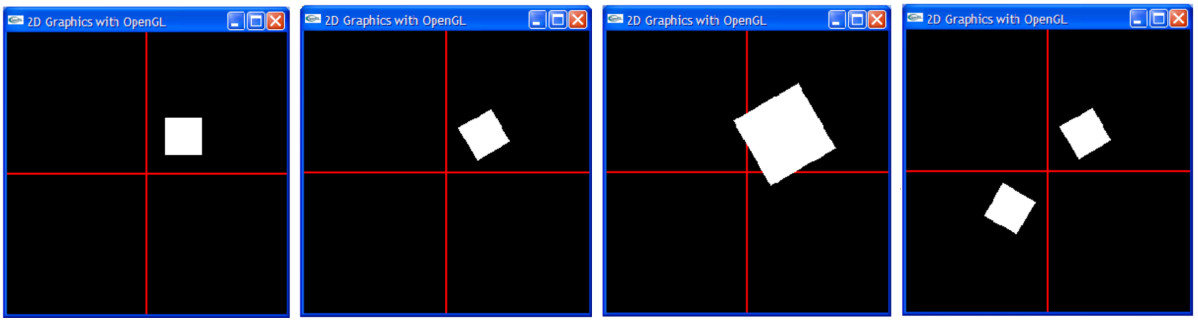


Figure 5. Drawing a quadrilateral and performing a sequence of geometric transformations with OpenGL.

2.8 Treatment of Colors

We begin this lecture by discussing how we perceive and how we quantify colors. We discuss other topics, such as the usual ways to represent colors [4], rasterizations, and take the opportunity to apply the knowledge learned so far, to advance further in OpenGL syntax.

2.9 Programming in a graphical event driven iterative environment

In this lecture, we visit again the paradigms of signals & slots of Qt, but now to design a graphical event driven iterative environment. We perform the implementation of a simple graphical program for two-dimensional visualization control. The program interface contains a toolbar at the top with the following actions: Fit, which adjusts the object's image on the program's canvas; Zoom in, which enlarges the object's image on the canvas; Zoom out, which reduces the object on the canvas; Pan left, which moves the object's image to the left; Pan right, which moves the object to the right; and so on. It is a good moment to propose a second Homework.

2.10 Mouse Events on canvas

In this lecture, we develop a simple Python program to understand mouse events on OpenGL canvas using Qt, and to teach how to implement a function to convert points from raster to world coordinates, see Fig. 6.

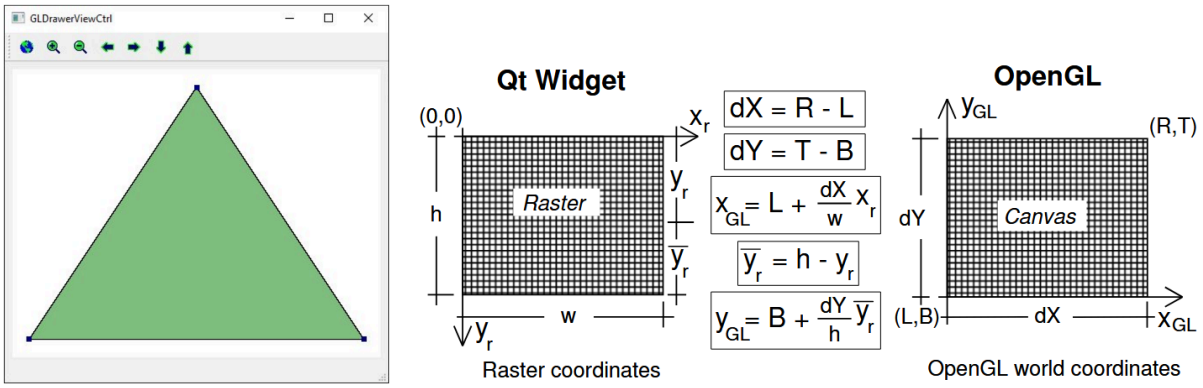


Figure 6. A graphical event driven iterative environment.

2.11 Digital Representations of curves

In this lecture, we discuss digital representations of curves. Four types of representations for curves and surfaces are common in Computer Graphics and Geometric Design: explicit, implicit, parametric, and procedural. Each of these alternatives will be briefly introduced, but only one particular form will be emphasized: parametric representation, which is used throughout the course.

2.12 Programming in a graphical environment for curve collections

In this lecture, we develop a program that handles the following curve types: Line: straight line segment; Polyline: polygonal line; QuadBezier: quadratic Bezier curve; CubicBezier: cubic Bezier curve; Circle: closed circular curve; and CircleArc: open circular arc curve. Only the Line curve is fully implemented. It is a good moment to propose the third Homework, in which the objective of this work is to add the processing (modeling and visualization) of other curve types to the program, see Fig. 7.

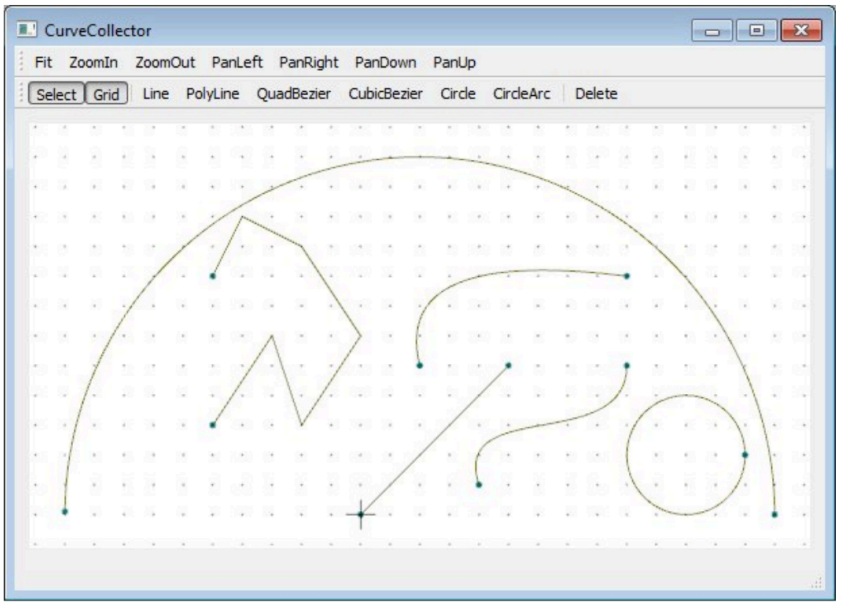


Figure 7. Curve collector.

2.13 Introduction to computational geometry

Computational Geometry is the study of algorithms for solving geometric problems on a computer. The emphasis in this course is on the design of such algorithms, with less attention paid to performance analysis [4]. In this lecture, we focus on predicates of computational geometry, precision, and exact and adaptive arithmetic.

2.14 Algorithms for Tessellation of 2d regions

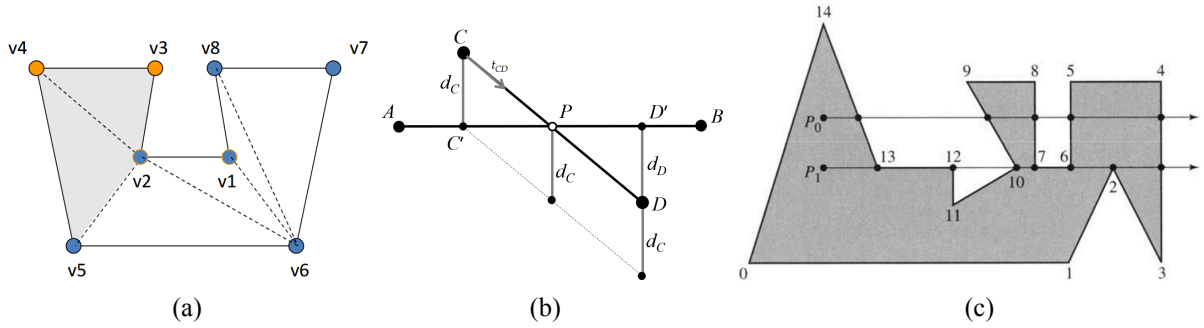
In this lecture, we show how to implement algorithms to tessellate non convex planar regions. The basic idea is to search for the triangle/ears, see Figure 8. Any polygon with more than three sides has at least two ears. An ear is defined if the angle between the edges emerging from the vertex is less than 180° and the chord connecting the two adjacent vertices must not intersect any other edge of the polygon (i.e., no other vertex must lie in the triangle/ear), see Fig. 8(a).

2.15 Algorithms for line-line intersections

Testing whether a given pair of segments intersects proves surprisingly tricky because of difficult special cases that arise. Two segments may lie on parallel lines, meaning they don't intersect at all. One segment may intersect the end of another, or the two segments may be on top of each other so that they intersect at a segment rather than a single point. We show how to implement an algorithm to treat the intersection of line segments in a robust and efficient way using predicates of computational geometry, see Fig. 8(b).

2.16 Proximity tests e points inclusion

A ray that starts from any point inside a polygon in any direction will intersect the curves on the polygon's boundary an odd number of times. If the ray intersects the polygon's boundary an even number of times, the point is outside the polygon, see Fig. 8(c). Finally, we establish a criteria for counting intersections of a ray with an edge segment. It is a good moment to propose the fourth Homework. At this point, we have developed full capability to implement a naive geometric modeler with curve intersections and explicit region creation.



(a) (b) (c)

Figure 8. Algorithms for (a) tessellation, (b) segments intersection, and (c) point inclusion.

2.17 Geometric modeling (2-manifold and non-manifold)

We begin this lecture discussing the difference between graphic modeler and geometric modeler. Until now, we have developed all capabilities to produce a graphic modeler, but now we need to advance further and develop the skills to build a geometric modeler. We introduce the idea of 2-manifold and non-manifold solids and the need to deal with topology in addition to geometry.

2.18 Topological data structures

In this lecture, we dive into topological data structures for solid modeling, and introduce the well known Winged-edge topological data structure, doing several exercises, including with the Euler operators.

2.19 Half-edge data structure

In this lecture, we introduce the half-edge topological data structure, and the notion of abstractions such as half-edges and loops, to perform advanced and general planar subdivisions. Finally, we have all we need to create our geometric modeler, see Fig. 9. It is a good moment to propose the fifth Homework.

2.20 Finite element mesh generation

In this lecture, we exhibit the world of mesh generation with focus on finite elements. We discuss structured and unstructured meshes and the need for specialized algorithms.

2.21 Mapping algorithms for mesh generation

We implement an algorithm based on transfinite interpolation to generate quadrilateral structured meshes in this lecture, see Fig. 9.

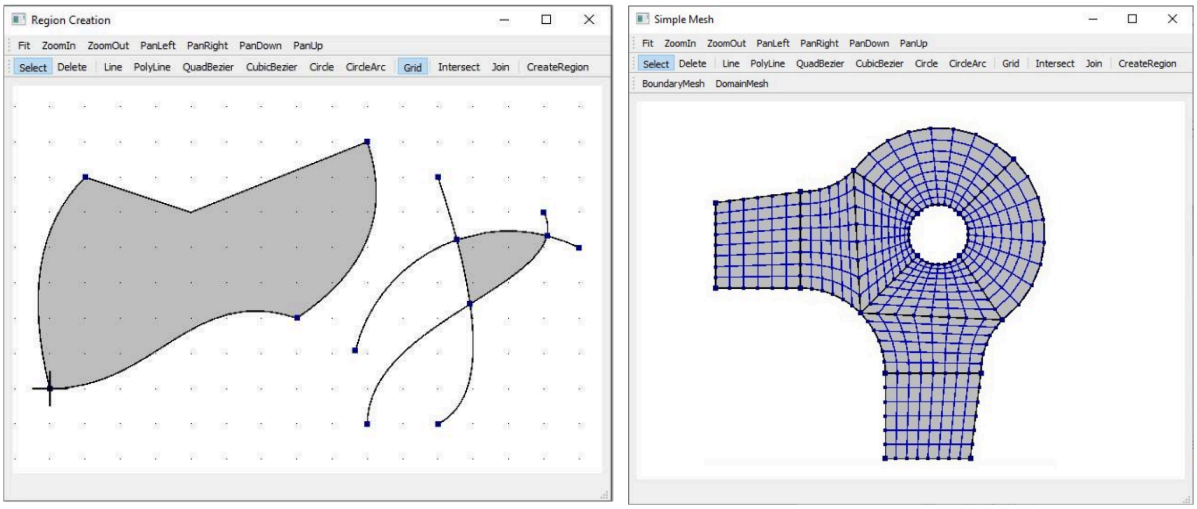


Figure 9. Geometric modeler for planar subdivision (left) and quadrilateral mesh generator with mapping algorithms (right).

2.22 Advancing front and Delaunay triangulation

In this lecture, we start to approach unstructured mesh algorithms. Two good method examples are the advancing front and Delaunay triangulation

2.23 Managing modelling attributes: computational simulation with finite elements

In this lecture, it is time to put it all together. The only detail missing, but without it we cannot perform computational simulations, are the attributes, i.e., the icing on the cake. Although it is not as complicated as other topics, it is underestimated, so it is always a challenge to efficiently implement a system capable of managing attributes effectively and in a versatile way. Finally, we present a generic 2D modeler with mesh generation.

2.24 Introduction to scientific visualization

In this final lecture, we discuss the difference between data visualization and scientific visualization. We work on the implementation of 3D geometric transformations in homogeneous coordinates and mechanisms for visualization control based on screen axes, see Fig. 10. It is time to propose the sixth and last Homework.

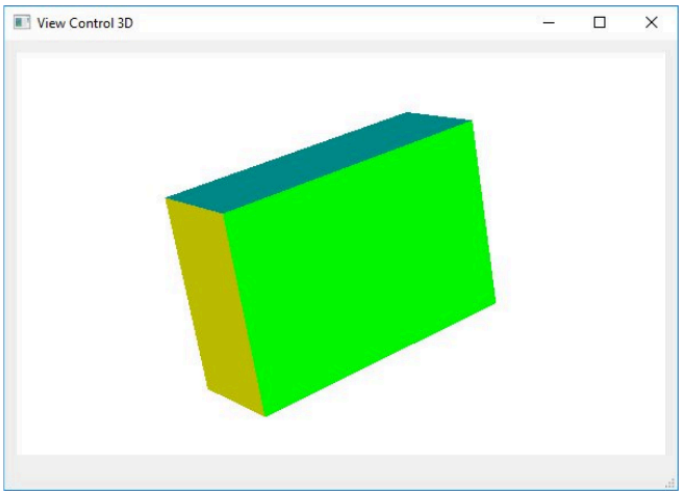


Figure 10. 3D scientific visualization with OpenGL.

3 Conclusions

In this paper, we described practical experiences, acquired over several years, in the development of a graduate level course in applied computer graphics. Since the target audience of the course is engineers and scientists, we focus mainly on topics related to pre-processing and post-processing techniques, which, although fundamental for performing numerical simulations, are usually neglected in the educational process.

Our main challenge is to teach very complex subjects that students will be exposed to for the first time in this course. However, students who pursue such courses are usually at a high level of maturity and understand that the topics covered are fundamental for their career and research. Therefore, to give an effective and practical character to the course, we follow an approach that is completely based on the development of practical software with user interfaces, relying on open source packages with Python APIs. We chose to use Qt and OpenGL to create interactive graphical user interfaces in a hands-on scheme.

Accompanying the students, we have received very positive feedback, realizing that the course is crucial for building a solid foundation for their careers. It is notorious that they are capable of benefiting from and applying the knowledge taught in their research work.

Acknowledgements. This research was supported by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Finance Code 001, the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), grant no. 308884/2021-3, and the Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ), grant no. E-26/201.224/2022.

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] Foley, J.D., van Dam, A., Feiner, S., Hughes, J.; Computer Graphics: Principles and Practice, Second Edition in C, Addison-Wesley, 1995.
[2] Waldemar Celes, Renato Cerqueira e José Lucas Rangel, Introdução à Estrutura de Dados – Com técnicas de programação em C, 2ª edição, Editora GEN LTC, 2022.
[3] Rogers, D.F., Adams, J.A.; Mathematical Elements for Computer Graphics, Second Edition, McGraw-Hill International Editions, Computer Series, New York, 1990.
[4] Jonas Gomes e Luiz Velho, Computação Gráfica, Volume 1, Série de Computação e Matemática, IMPA, 1998.