Computer Graphics and Visualization

Practice Nº 1

3D visualization

Contents:

- Basic primitives.
- Geometric transformations, matrix composition, matrix stack.
- The synthetic camera, projection, viewport.

Objectives:

- Place different primitives on a scene and apply geometric transformations.
- Place a camera on the scene and implement different camera movements and parameter changes.

Number of lab sessions to complete the practical: 4 sessions (8 hours).

Score: 3.5 out of 10, then weighted at 60% (practical grade).

Explanation:
During two of the practical sessions, at the beginning of the sessions, the teaching staff will explain the fundamental concepts for carrying out the practical work if they have not yet been covered in theory, based on the code solved in the statements developed at the end of this document.

Based on this code, implemented using C++ classes that utilize the OpenGL library and the development environment installed in the teaching laboratories, it is necessary to develop a program that solves the problem outlined in this practical exercise on 3D visualization.

Three different objects must be generated, consisting of several parts and implemented with basic primitives such as cubes, cylinders, cones, spheres, etc. These objects must be displayed as part of a scene.

The generated objects must be selectable using the keyboard ("1," "2," "3") and, once selected, allow sequential transformations (translation, rotation, and scaling) to be applied using the keyboard, so that the transformations accumulate:

Translation in X: **"left cursor"** (positive) and **"right"** (negative)
Translation in Z: **"up cursor"** (positive) and **"down"** (negative)
Translation in Y: **"u"** (positive) and **"U"** (negative)
Rotation in X: **"x"** (positive) and **"X"** (negative)
Rotation in Y: **"y"** (positive) and **"Y"** (negative)
Rotation in Z: **"z"** (positive) and **"Z"** (negative)
Homogeneous scaling in XYZ: **"s"** (increase) and **"S"** (decrease)

For the above transformations, two operating modes will be used: one that allows the transformations to be applied by always accumulating all **rotations**, **scalings** and **translations** independently and applying them in the order R-S-T based on the coordinate

origin and another that allows the sequence in the order of pressing, i.e., that allows different types of transformations to be intercalated. To change modes, press the **"m/M"** key.

To visualize the above, a camera had to be used in a specific position. Interactive camera movement must also be enabled using the keyboard. Specifically, the camera must be able to orbit around the coordinate origin, pitch, and rotate around the Y axis located on the camera itself. To do this, these functionalities must be implemented in the Camera class, including the appropriate variables and functions for these types of movement.

The mode of operation will be as follows: after activating the camera movement mode by pressing **"c/C",** the left, right, up and down cursors will be used for orbit and pitch movements, and the **"y/Y"** key will be used for rotation on its axis.

The camera will also allow the scene to be cropped, including this functionality in the camera class, so that the front and rear planes of the viewing volume can be moved forward and backward using the **"f/F"** keys for the front plane and **"b/B"** for the rear plane. Zooming will be performed using the **"+/-"** keys.

Finally, several viewports will be implemented to allow the scene to be viewed from different angles, as well as to switch between parallel and perspective projection using the **"p/P"** key, all by modifying the Camera class appropriately.

Practice summary:

- Generate three different objects based on basic primitives.
- Select and object using keyboard and apply a sequence of transformations interactively using the keyboard.
- Place a camera in the scene and move it using the keyboard with different types of camera movements, modifying its different parameters.
- Perform cropping and zooming operations.
- Implement several simultaneous viewports.

Practical assignment submission:

- Self-documenting C++ code, using the classes provided.
- Explanatory video showing the application in operation.
- Explanation of fundamental details of the implementation.

Submission of the assignment:

Upload a **ZIP** file containing only **source code files** (.cpp and .h). Assignments must be submitted on time in order to be evaluated.

Explanatory example for Geometric Transformations

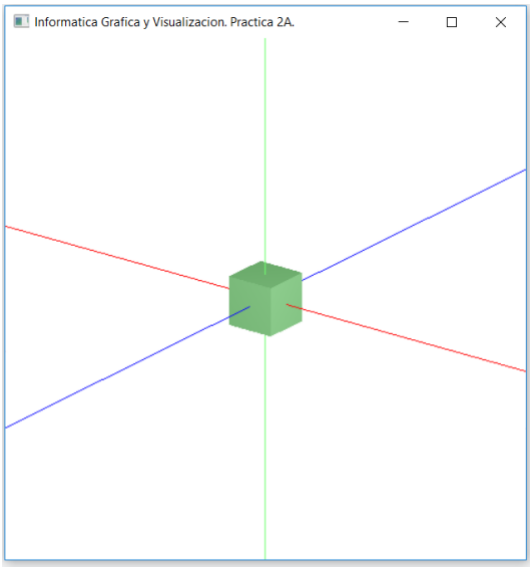
Objectives:

- Practice with the concept of transformation matrices.
- Apply transformations: translation, scaling, and rotation.
- Use matrix composition.
- Combine multiple transformations using the matrix stack (required).

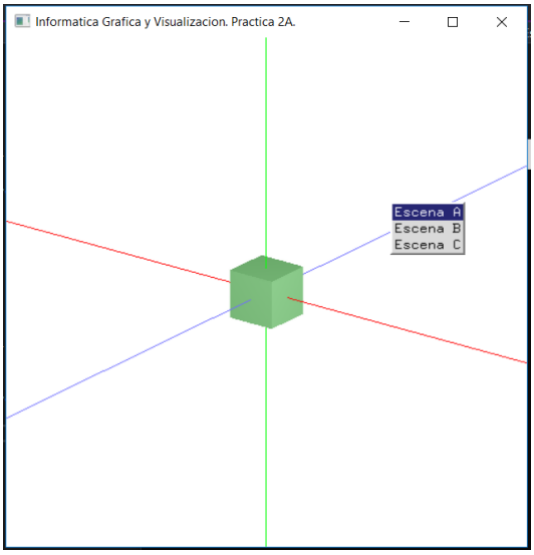
Part of the code:

- **pr2a.cpp**: main() function of the program
- **igvInterfaz.h and igvInterfaz.cpp:** specification and implementation of the igvInterfaz class, which contains the basic functionality for creating a display window, its configuration, and the management of system events.
- **igvEscena3D.h and igvEscena3D.cpp:** specification and implementation of the igvEscena3D class, which contains the basic functionality for displaying a scene.

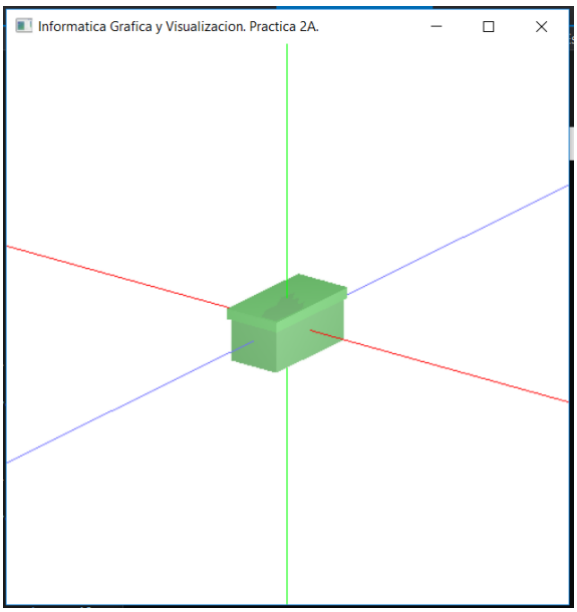
If we run the program without making any changes, a window opens showing the following:



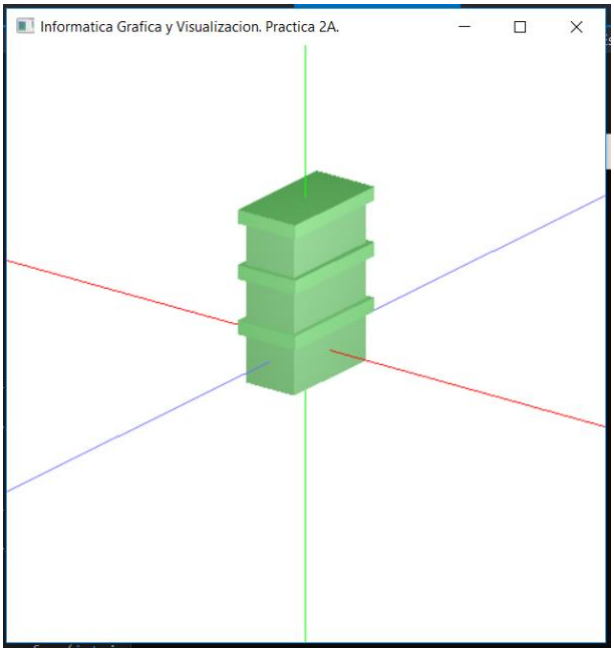
The program includes a menu for changing scenes (there is one created for each exercise). To display this menu, right-click on the window.



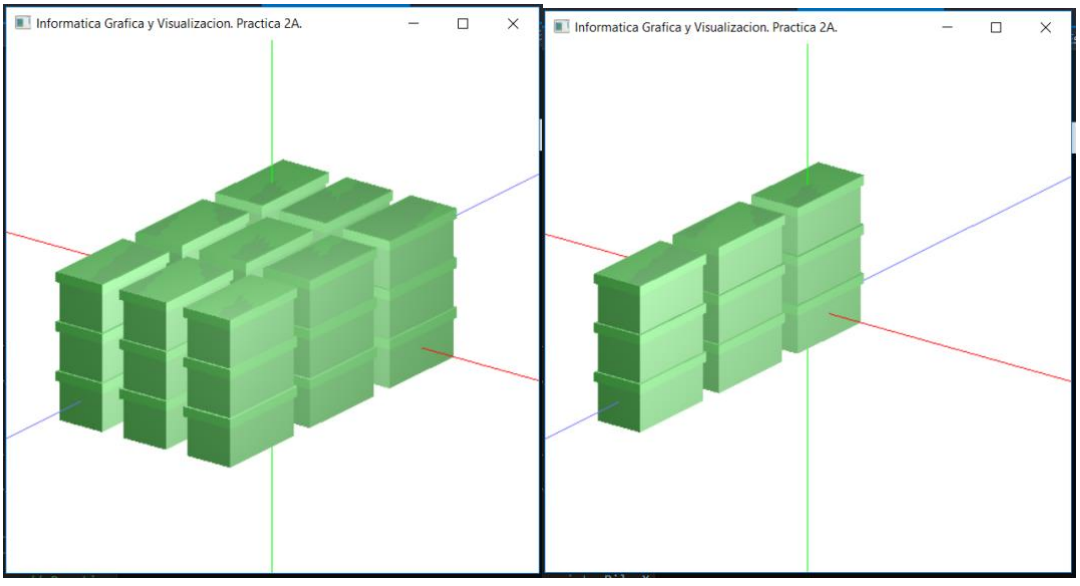
a. Add the necessary code to the igvEscena3D::renderEscenaA() method to display the following figure (use of the matrix stack is mandatory):



b. Add the necessary code in the igvEscena3D::renderEscenaB() method to build a stack with instances of the model in section a. The number of instances must be parameterizable. If section c is not performed, pressing the “y”/“Y” key must be implemented as described in that section.



c. Modify the code of the igvEscena3D::renderEscenaC() method to interactively replicate the stack of instances along the X and Z axes. This allows you to increase/decrease the number of instances in each stack by pressing the corresponding keys: “x”/“X” to increase/decrease the number of stacks of boxes on the X axis, “z”/“Z” on the Z axis, and ‘y’/“Y” to increase/decrease the number of boxes in the stack.



Explanatory example for

the vision camera, projections, and viewport

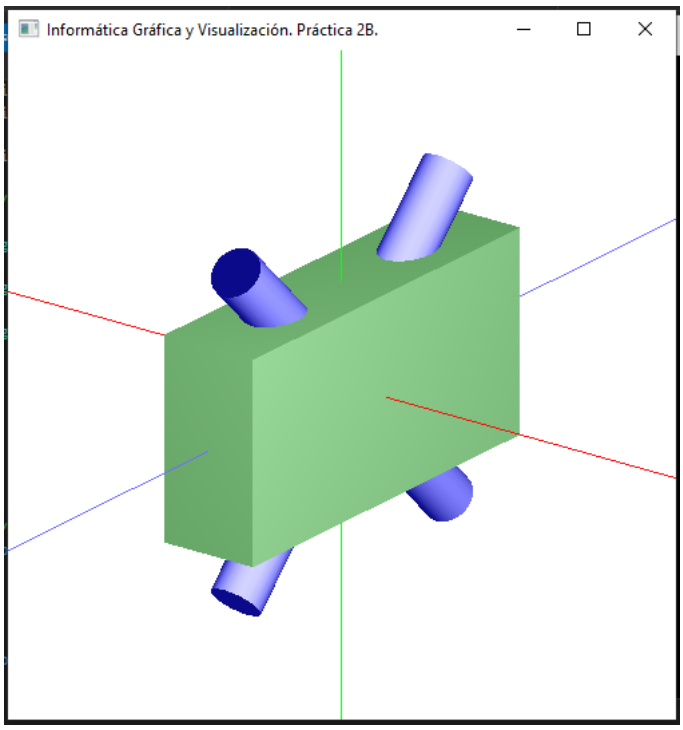
Objectives:

- Understand camera behavior. Define and properly use parameters for positioning and orienting the vision camera.
- Reinforce concepts related to parallel and perspective projections.
- Use viewport transformation.

Part of the code:

- **pr2b.cpp**: main() function of the program
- **igvInterfaz.h** and **igvInterfaz.cpp**: specification and implementation of the igvInterfaz class, which contains the basic functionality for creating a display window, its configuration, and the management of system events.
- **igvEscena3D.h** and **igvEscena3D.cpp**: specification and implementation of the **igvEscena3D** class, which contains the basic functionality for displaying a scene.
- **igvPunto3D.h** and **igvPunto3D.cpp**: specification and implementation of the igvPunto3D class, which contains the functionality to declare and use point and vector objects.
- **igvCamera.h and igvCamera.cpp**: specification and implementation of the igvCamera class, which contains the basic functionality for creating and manipulating cameras in the application.

If we run the program without making any changes, a window opens showing the following:

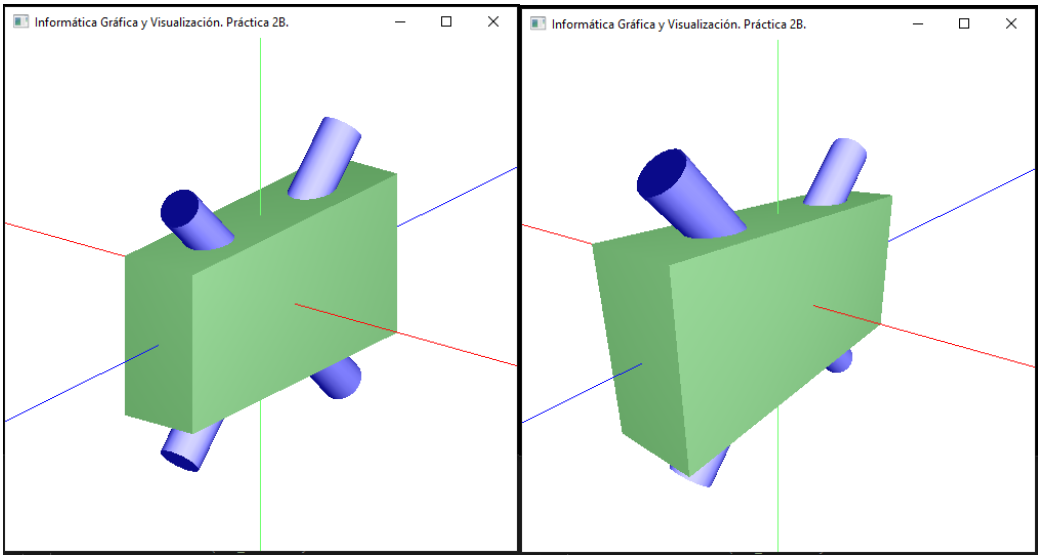


- The application displays an object centered on the coordinate origin consisting of a parallelepiped with sides 1, 2, and 4, and two cylinders with radius 0.25 and length 4.
- It uses a parallel projection with the camera parameters:

- Viewpoint: (3, 2, 4)
- Reference point: (0, 0, 0)
- Upward vector: (0, 1, 0)

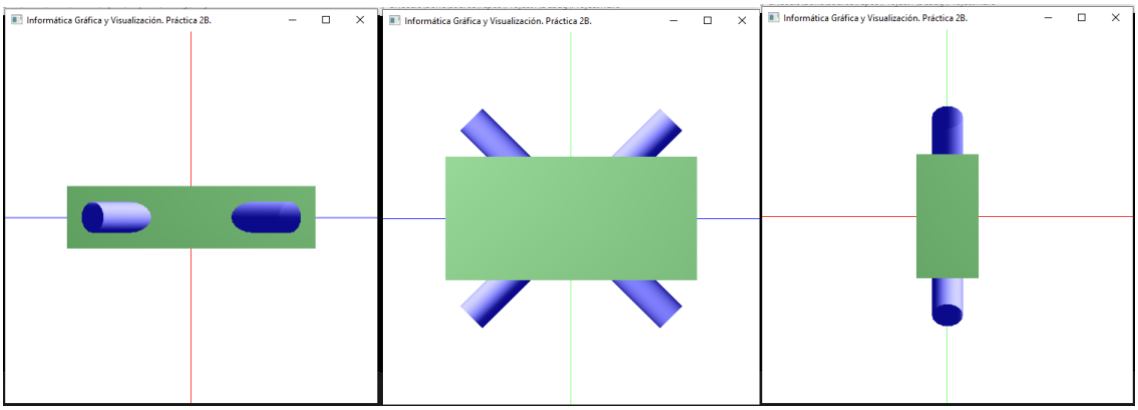
- The ‘e’ key activates/deactivates the display of the coordinate axes.
- The ‘escape’ key terminates the program.

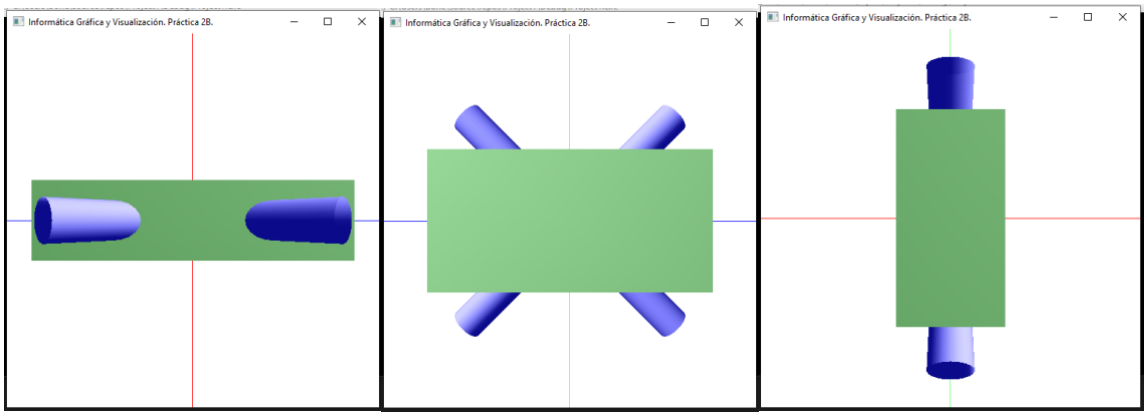
a. Make the necessary modifications to the code in the igvInterfaz::keyboardFunc method so that pressing the ‘P/p’ key switches the projection type from parallel to perspective (angle and aspect) and vice versa. The camera position must remain the same in both projections:



b. In addition to the original panoramic view, new views of the model must be added: plan (viewpoint on the Y axis), profile (viewpoint on the X axis), and elevation (viewpoint on the Z axis). Change the initialization of the igvInterfaz::camera attribute to represent these additional views.
    a. Modify the igvInterfaz::keyboardFunc method so that you can interactively change the model views by pressing the ‘V/v’ key.
    b. b. If you are in a projection type and press the ‘P/p’ key (section A), you should obtain the same view, but changing the projection type.

The different views for each type of projection should appear as shown in the following figures (the initial panoramic view will correspond to the figure in section A):

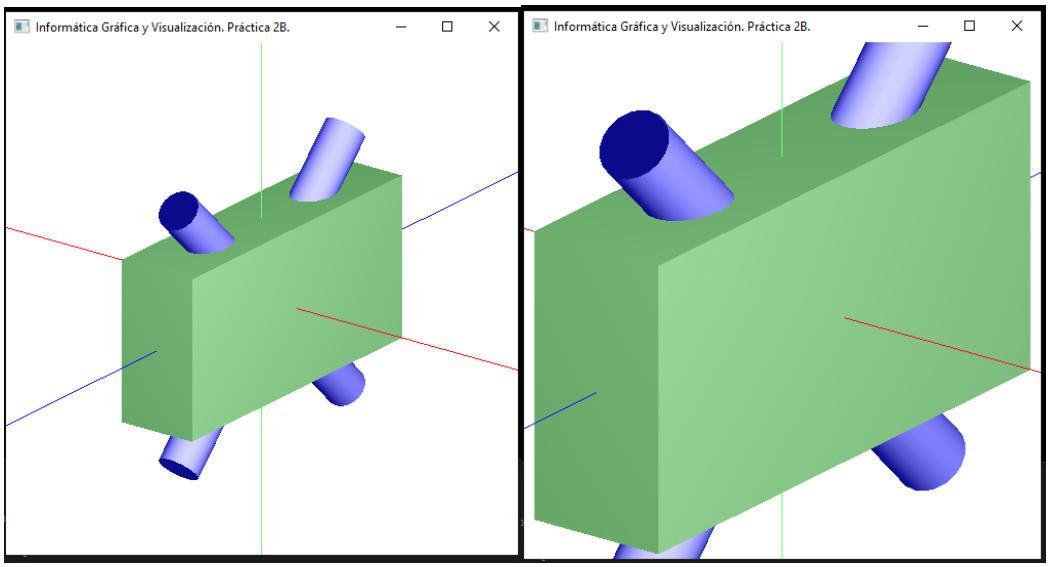


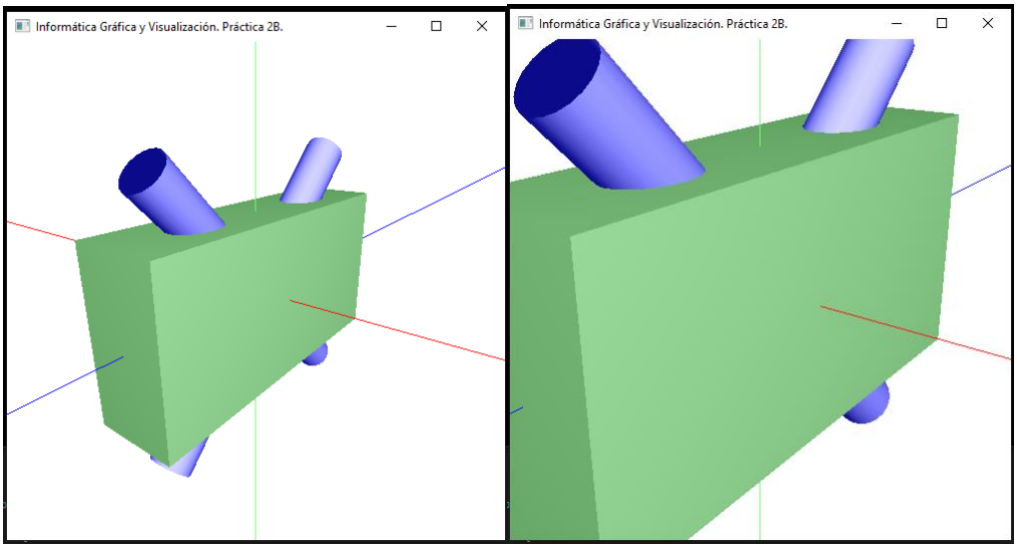


c. Implement the void igvCamera::zoom(double factor) method to zoom the camera.
    a. The zoom must be implemented in such a way that the camera parameters are modified so that the image size zooms in or out depending on the value of the factor parameter (zoom percentage).
    b. The camera position must not be modified when zooming.
    c. The zoom must work in both parallel and perspective projection.
    d. The coordinates of the object must not be modified when zooming.

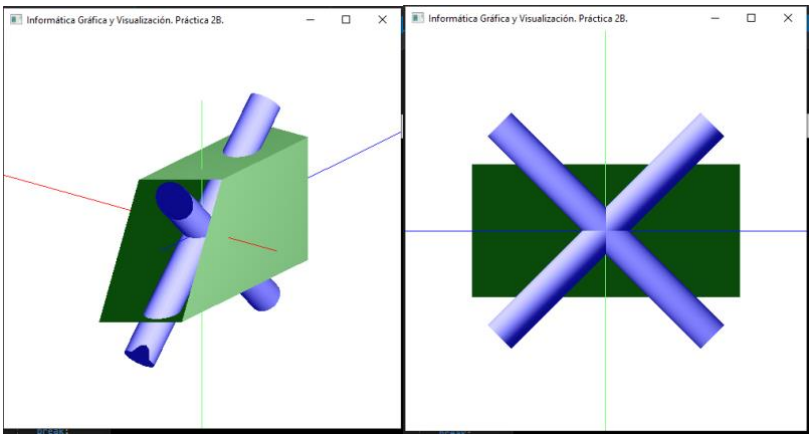
Add the necessary code to the igvInterfaz::keyboardFunc and igvCamara::zoom(double factor) methods to interactively change the zoom percentage using the '+' (zoom in) and '-' (zoom out) keys. Each press of these keys will result in a 5% zoom factor (increase or decrease).

For example, if you press the '+' key 10 times, you should see the display shown in the following figure for parallel and perspective projections. Pressing the '-' key 10 times should return you to the starting position:

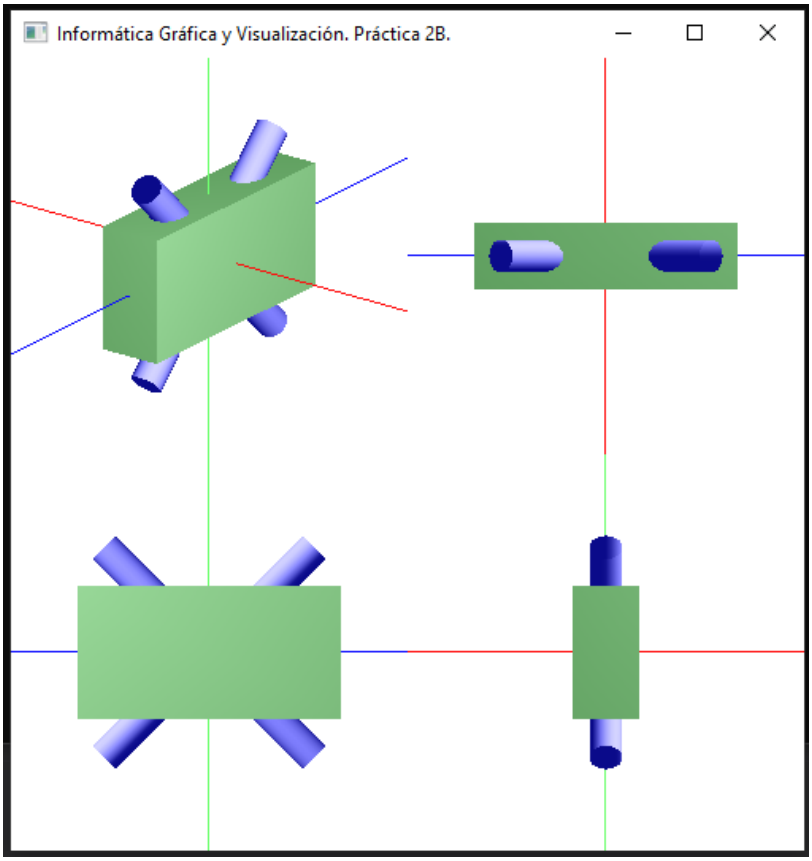




d. Add the necessary code in the igvInterfaz::keyboardFunc method to interactively change the distance of the near plane of the viewing volume defined by the camera (both in parallel and perspective projection). Each press of the 'n' key will increase the value of the near plane distance by 0.2 units, and pressing the 'N' key will decrease the value by the same amount. After pressing the 'n' key several times, the near plane should pass through the model and display visualizations such as the following (parallel panoramic projection and perspective projection in plan):



e. Add the necessary code to the igvInterfaz::keyboardFunc method and modify the igvInterfaz::displayFunc method and the igvInterfaz class to divide the window into four views by pressing the '4' key, taking into account the following restrictions:
    a. The camera settings should not be changed.
    b. Pressing the '4' key again will restore the original format.
    c. All the features of this practice should work with the 4 views.
    d. This configuration must be adapted to the screen size.


Informática Gráfica y Visualización. Práctica 2B.