Transforming JDOR into JDORFX: Providing 3D Graphics to ooRexx

Philipp Schaller

01125001

Department of Information Systems & Society

Reviewer: ao.Univ.Prof.Mag.Dr.rer.soc.oec Rony G. Flatscher

26th of May 2024

Table of Contents

List of Abbreviations

GUI
: Graphical User Interface.

awt
: Abstract Window Toolkit

List of Figures

List of Tables

List of Listings

Abstract

This thesis introduces JDORFX, a JavaFX-based graphics framework, which utilizes the capabilities of the Bean Scripting Framework (BSF) for ooRexx to provide JavaFX graphics classes to ooRexx programmers. Based on the JDOR framework, which leverages awt based Java2D for graphics rendering, it now also offers the use of 3D classes. The development process of JDORFX is described, highlighting architectural differences to JDOR. Both frameworks are compared to detect potential differences in performance and output. Finally, nutshell examples are provided to explain each ooRexx command and their arguments that utilizes JavaFX 3D classes through JDORFX.

1. Introduction

JDOR (Java Drawing for ooRexx) is a BSF4ooRexx850 extension that facilitates awt based Java2D classes to the ooRexx programming language. This means that ooRexx programmers can utilize the rich set of tools and functions available in these Java classes and create and manipulate 2D graphics and images within their ooRexx scripts [1].

JavaFX is a Java library developed to replace Java swing and awt based Java2D as GUI frameworks. It is used to develop versatile applications, encompassing both desktop and Rich Internet Applications. These JavaFX applications are capable of seamless execution across multiple platforms. It features more functionalities than swing or awt based Java2D, such as Java's 3D classes [2].

The goal of this Bachelor thesis is 1) to develop JDORFX, a BSF4ooRexx850 extension that implements the same capabilities as JDOR, but transitioning from awt based Java2D classes to JavaFX and 2) expanding it with the capabilities of Java3D. This will allow ooRexx programmers to seamlessly switch between the two frameworks using the same syntax within 2D graphic creation, but also facilitate an interface for 3D graphic creation to the ooRexx language.

2. Background

This chapter provides an overview of the programming languages and libraries utilized in the development of JDORFX, describing their features and functionalities. These components collectively enable the integration, scripting, and graphical rendering capabilities of the framework.

2.1. Open Object Rexx

Open Object Rexx (ooRexx) is an open-source implementation of Object Rexx managed by the Rexx Language Association (RexxLA) and distributed under the Common Public License. It builds on classic Rexx, which was developed in 1979 by Mike F. Cowlishaw and IBM [3]. While compatible with classic Rexx and retaining its ability to be written procedurally, it extends the capabilities of classic Rexx with object-oriented features such as subclassing, polymorphism, and data encapsulation. One key advantage of ooRexx is its user-friendly nature, with its syntax derived from meaningful English words for instructions and little formatting requirements, which allows instructions to span multiple lines and be written in upper or lower case. Further, ooRexx treats all data as objects, eliminating the need to declare variables as specific types and permitting arithmetic operations on strings which represent valid numbers. Rexx covers a variety of functionalities which are usually offered by fundamentally different types of programming languages, including the ability to develop programs of varying complexity, tailored user commands without being dependent on its primary environment of operating systems, providing a macro language for various applications, and developing prototype applications [4].

2.2. Java

Java, developed by Sun Microsystems in 1995, is an object-oriented programming language as well, utilizing objects to represent both data and functionality [5]. It implements concepts such as classes, inheritance, polymorphism, abstraction and encapsulation, which are provided by its extensive standard library (API). Java is known for its simplicity, security, and robustness. One of its key features is that it is a software based platform, which makes it platform independent and capable of being executed on multiple platforms. Security is another highlight, with Java running on a virtual machine, using a classloader, which separates local class packages from imported ones, a bytecode verifier, which scans for illegal code, a security manager that governs classes' access rights, and not implementing explicit pointers. Strong memory management, automatic garbage collection of unused objects and exception handling offer robust mechanisms. Its architecture neutrality further prevents implementation dependencies [6].

2.3. Bean Scripting Framework for ooRexx

The Bean Scripting Framework (BSF) for ooRexx is based on the open-source BSF (Bean Scripting Framework) class library of Java, designed to facilitate scripting language integration within Java applications, enabling access to Java objects and methods [7]. BSF4ooRexx provides interoperability between ooRexx and Java, allowing ooRexx scripts to utilize Java classes and libraries seamlessly and enabling compatibility across various operating systems and environments. It synthesizes ooRexx's human-oriented design principle and Java's extensive capabilities by camouflaging Java objects as ooRexx objects. Its latest version is BSF4ooRexx850 and requires a Java version of at least 8 and an ooRexx version of 5.0 as minimum requirements. Embedded within BSF4ooRexx850 is JDOR.

2.4. JDOR

JDOR (Java Drawing for ooRexx) serves as a powerful tool for ooRexx programmers, enabling the utilization of awt based Java2D classes for graphic creation and manipulation

[1]. It serves as a Rexx command handler, a runtime library written in Java with BSF4ooRexx850. The primary goal is to facilitate smooth interfacing with the awt based Java2D subsystem, while still applying easy to use ooRexx syntax.

It operates by instantiating a Java class, "JavaDrawingHandler", which implements the "RexxRedirectingHandler" to process redirected ooRexx commands and executing them through the "handleCommand" callback method. Each command is translated into Java code while being processed in the "processCommand" method. Upon receiving the first command, the handler instantiates a "JFrame" [8] containing a "JPanel" [9]. The "Graphics2D" [10] class allows to either draw graphic components directly onto a "BufferedImage" [11] or to create shape [12] and string objects which can later be added.

2.5. JavaFX

JavaFX [2], in contrast to Java's Swing classes, consists of graphics and media packages facilitating the development of rich client applications. Their capabilities consist of Java APIs, FXML for creating JavaFX application user interface, WebView, and Swing interoperability, which allows to implement JavaFX capabilities to existing Swing applications. It also offers built-in UI controls and CSS, 3D graphics features, a Canvas API to draw within scenes, printing functionality, Rich Text Support enhances text capabilities, and multitouch support caters to modern input methods. JavaFX further supports smooth graphics rendering and a high-performance media engine for stable web multimedia content.

Its architecture [2] consists of several interconnected components starting with the scene graph which serves as the foundation for building JavaFX applications, representing a hierarchical tree of nodes, starting with its root node. All other visual elements have exactly one parent node and zero or more child nodes. Nodes can be objects such as 3D shapes, 2D shapes, cameras, lights etc. Aside from nodes, the scene graph can also build and store states, such as transforms, or effects, which can be applied to objects of the graph. The Java public APIs offer extensive support for rich client application development, leveraging Java features like generics and annotations. The graphics system, comprising Prism and Quantum Toolkit, facilitates rendering and event handling, ensuring smooth performance across platforms. The Glass windowing toolkit manages native operating system services and event queues, facilitating seamless integration with the JavaFX platform. Threads like the JavaFX application thread and Prism render thread handle various aspects of application execution and rendering. The media and images features allow support for audio and video. The web component, based on WebKit, offers functionalities such as HTML rendering, JavaScript execution and more. CSS facilitates customizable styling of UI elements, allowing developers to dynamically change the application's appearance. UI controls, layout containers, and transformations offer options for organizing and modifying the user interface. JavaFX further allows to implement visual effects to further enhance the visual appearance of JavaFX applications.

3. Requirements

This chapter describes the installation process of the programs that have been used for this project and are required to run the nutshell examples with JDORFX implementation. It provides instructions for windows, however installation guides for mac or Linux can also be found in the provided links. It is recommended to install the programming languages ooRexx and Java before BSF4ooRexx to prevent possible issues.

3.1. Java

Java 8 is the earliest version of Java that is still receiving updates by vendors [13] and is supported by BSF4ooRexx850. While it includes all the essential packages for development, it is worth noting that later versions of Java which implement JavaFX GUI modules can be utilized as well. Java can be downloaded from different sources. The provided link guides to the download page of bellsoft:

https://bell-sw.com/pages/downloads/#jdk-8-lts [14]

Fig. 1. shows the version of Java used for development in this thesis: Liberica Full JDK 8u402+7 x86 64 for Windows. It is essential to choose the Full JDK package with 64-bit rate, this will include the necessary JavaFX modules. Clicking on MSI downloads the installer.

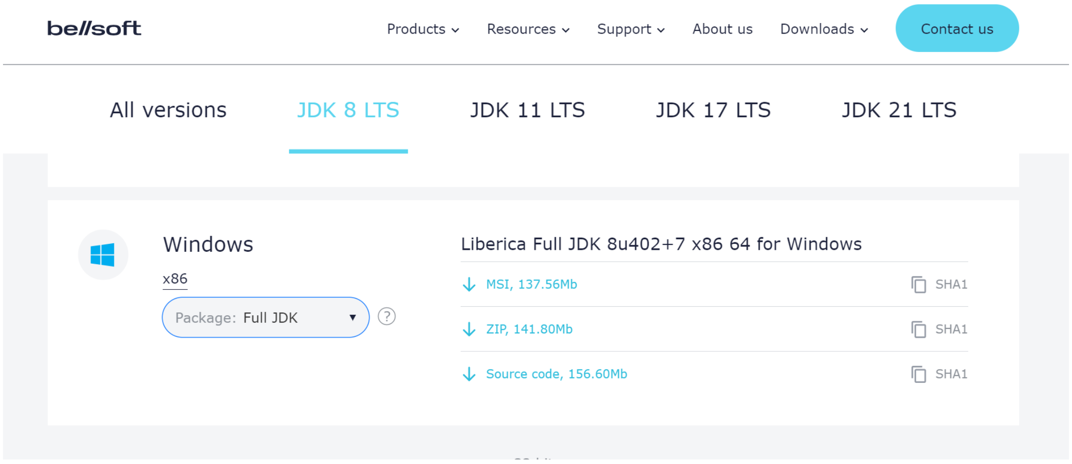

Fig. 1. Download Java [14]

Executing the msi file starts the setup, which lets the user select the features to be installed and their storage location. The default options (entire package and the path "Programm Files") should not be changed and the installation can be started.

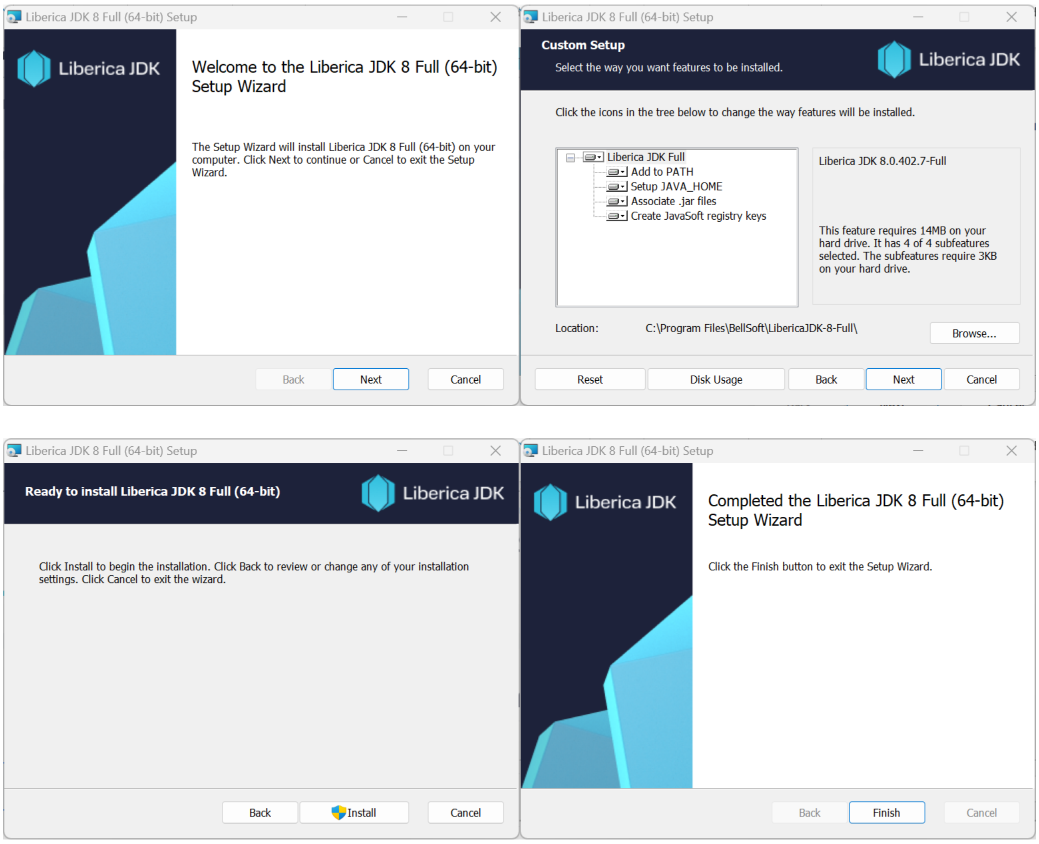

Fig. 2. Java Installation

After the process is finished, one may open a windows terminal and enter the command "java -version", which shows the currently installed version of Java on the operating system. Its number should match the one downloaded.

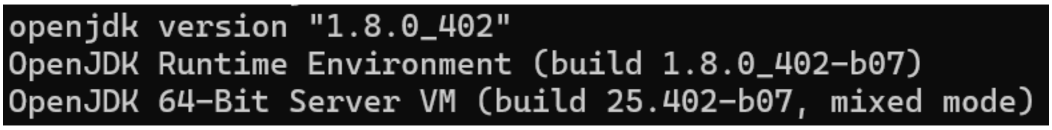

Fig. 3. Terminal Command "java -version"

3.2. Open Object Rexx 5.0.0

The required version of ooRexx is 5.0.0 or later with a bit rate of 64, which can be downloaded from sourceforge via the following link:

[https://sourceforge.net/projects/oorexx/files/oorexx/5.0.0/ooRexx-5.0.0-12583.windows.x86_64.exe/download [15]](https://sourceforge.net/projects/oorexx/files/oorexx/5.0.0/ooRexx-5.0.0-12583.windows.x86_64.exe/download)

The download will start automatically after 5 seconds or if the download button is clicked. Once completed, users will need to open their download directory, right click on the downloaded file and choose “Properties” (German: Eigenschaften). In the “General” tab (German: Allgemein), the checkbox “Unblock” (German: Zulassen) must be checked and confirmed by clicking “Apply” (German: Übernehmen) and then “OK”.

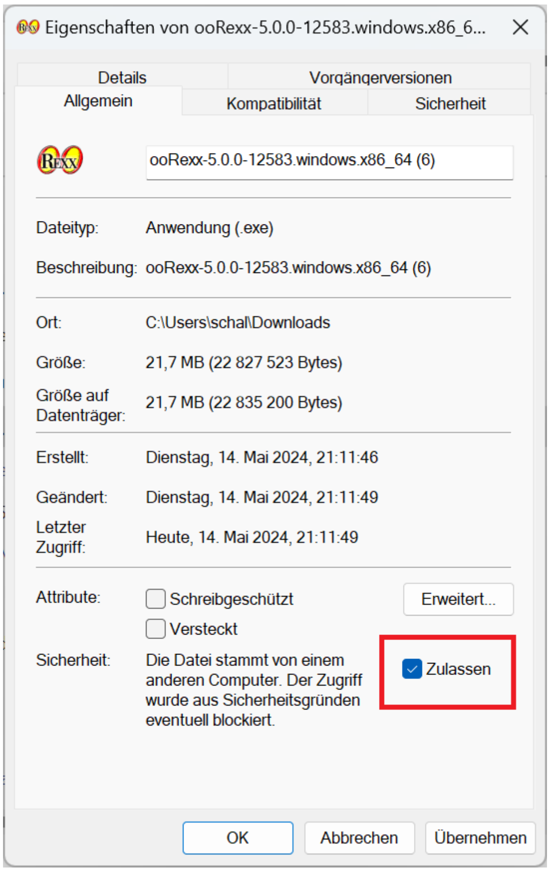

Fig. 4. Unblock ooRexx

Doing this will prevent Windows from blocking the installation process, a situation that might occur if the operating system perceives the downloaded file as a potential security risk. Next, the setup will be started when executing the exe file. If an older version of ooRexx is already installed, choices appear to either upgrade to the new version or uninstall the previous one. In order to prevent possible issues later on, it is advised to select the uninstall option. All further preferences should remain as default and the installation process can be started.

Fig. 5. ooRexx Installation Setup

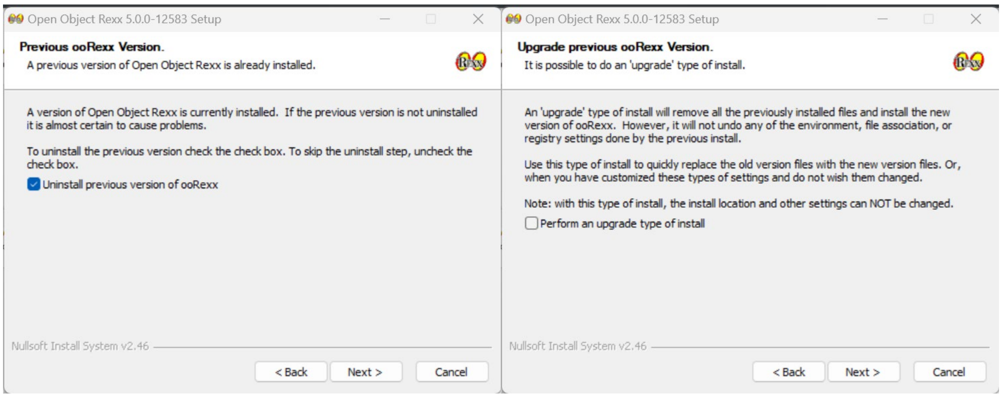

Fig. 6. Uninstall Older ooRexx Version or Upgrade

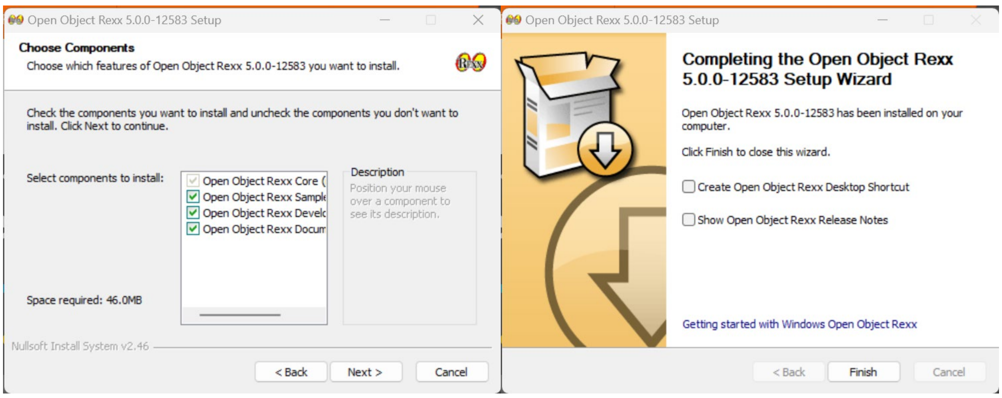

Fig. 7. ooRexx Installation Setup

Opening a terminal and entering “rexx -version” will check the installed version of ooRexx.

```
Open Object Rexx Version 5.0.0 r12583
Build date: Dec 23 2022
Addressing mode: 64
Copyright (c) 1995, 2004 IBM Corporation. All rights reserved.
Copyright (c) 2005-2022 Rexx Language Association. All rights reserved.
This program and the accompanying materials are made available under the terms
of the Common Public License v1.0 which accompanies this distribution or at
https://www.oorexx.org/license.html
```

Fig. 8. Terminal Command "rexx -version"

3.3. BSF4ooRexx850

Now that the programming languages have been successfully installed, the Bean Scripting Framework for Open Object Rexx, the bridge between Java and ooRexx, can be installed. If an older version of BSF4ooRexx is already installed, then it is advised to uninstall it before proceeding. The following link from sourceforge will start the download of the latest version automatically after 5 seconds:

https://sourceforge.net/projects/bsf4oorexx/files/latest/download [16]

Once again, in the properties window of the downloaded zip file, which can be opened by right clicking on the file, the checkbox “Unblock” needs to be checked.

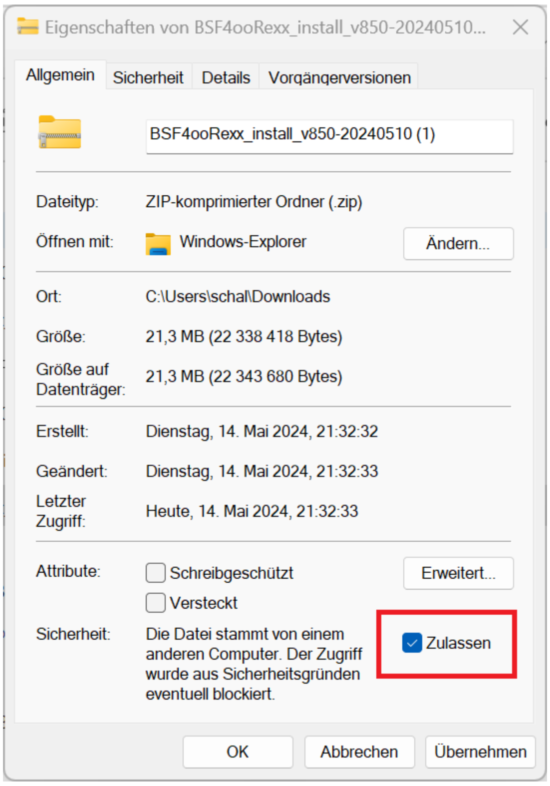

Fig. 9. Unblock BSF4ooRexx850

Next, the zip file needs to be unzipped. Within the newly created folder in "bsf4oorexx\install", the installation options for the different operating systems can be found. Opening the "windows" folder and executing the "install.cmd" file will open a windows terminal and start the installation process.

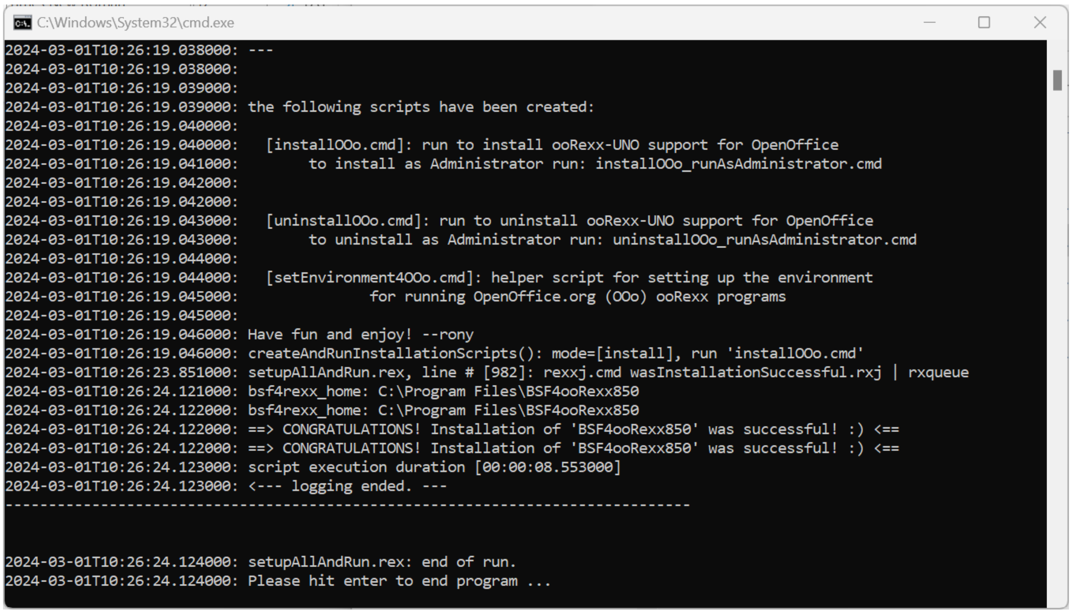

Fig. 10. BSF4ooRexx850 Installation

Finally, the file "ooRexxTry.rxj" in the folder "C:\Program Files\BSF4ooRexx850\utilities" can be executed to check whether all installations were done correctly and all three components are able to interact with each other. This opens a GUI window which accepts ooRexx commands as input.

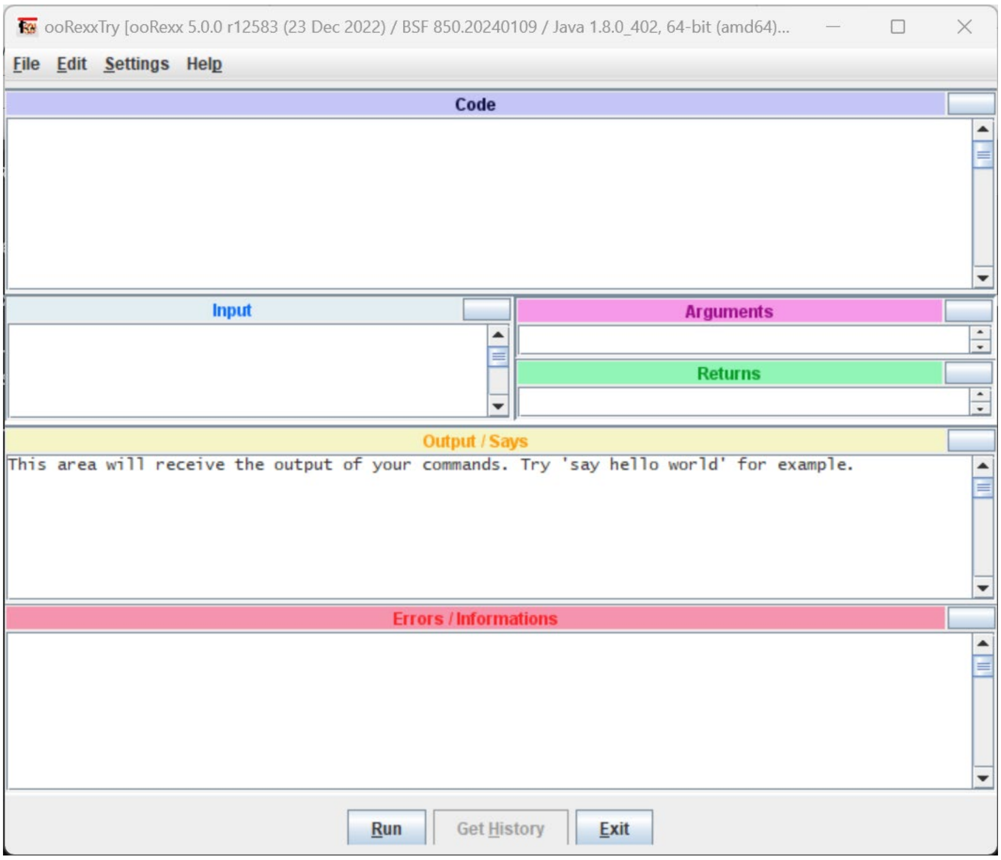

Fig. 11. ooRexxTry.rxj

4. Development

This chapter describes the development of JDORFX, using the previously described programming languages and packages.

4.1. JavaFXDrawingHandler

Building upon the foundation provided by the "JavaDrawingHandler", modifications have been made to utilize JavaFX packages instead of awt based Java2D. Hence, the "JavaFXDrawingHandler" class now extends the "Application" class [17] to incorporate JavaFX GUI functionality while still implementing the "RexxRedirectingCommandHandler". Within this class, the start method from the Application class has been implemented, which handles the top class container stage [18] and processes input and GUI updates. The stage class is the main window of an application and holds the scene graph, which can be switched during runtime.

When the "JavaFXDrawingHandler" class is instantiated, it iterates through the redirected ooRexx commands that have been supplied. Each input is processed via the “handleCommand” callback method, which in turn utilizes the “processCommand” method. Upon the first iteration, a new thread is started to invoke the application launch method, initializing the JavaFXApplication thread, which operates concurrently with the main thread. The Application class serves as the foundation for JavaFX applications. It initializes through the “init()” method which creates the GUI and updates it via the “start(Stage)” method [18]. The application stops running when either the “Platform.exit()” method is called or the last GUI window has been closed.

During the processing of the first command, a subclass of Scene [19], “JavaFXDrawingFrame”, is created. The JavaFX “scene” class acts as the container for a scene graph’s content. Its root Node determines how the scene graph adjusts to resizing.

Listining 1. shows the constructor of the “JavaFXDrawingFrame” class. Aside from static variables that define the scene’s size if no proportions were specified, a “Pane” [20] object serves as the root node, which enables dynamic adjustments of the scene's dimensions. Additionally, a “Canvas” for drawing functionalities, along with “Group” [21] nodes to store 2D shapes, 3D shapes, and “LightBase” objects are set as the root’s children. The method “resizeSurface” is implemented to change the scene’s size when needed.

```
class JavaFXDrawingFrame extends Scene
{
   /* static definitions          */
   static final private int prefWidth  = 500;
   static final private int prefHeight = 500;
   static final private boolean prefResizable=false;

   /* instance definitions         */
   boolean bDebug= false; // true
   Pane root = new Pane();
   Canvas canvas = new Canvas();
   Group shapeGroup = new Group();
   Group shape3DGroup = new Group();
   Group lightGroup = new Group();
   int currWidth  = prefWidth;
   int currHeight = prefHeight;
   boolean currFrameVisible = true;    // cf. command "winFrame [.true|.false]"
   boolean currFrameResizable = prefResizable;

   /** Constructor.
    * @param canvas to set size and display on scene
    */
   public JavaFXDrawingFrame(Canvas can)
```

```java
    {
        super(new Group());
        this.setRoot(root);
        currWidth = (int) can.getWidth();
        currHeight = (int) can.getHeight();
        root.setPrefSize(currWidth,currHeight);
        canvas = can;
        root.getChildren().add(canvas);
        root.getChildren().add(shapeGroup);
        root.getChildren().add(shape3DGroup);
        root.getChildren().add(lightGroup);
    }
    /** Resizes frame.
     *  @param width  the width in pixel
     *  @param height the width in pixel
     */
    public void resizeSurface (int width, int height)
    {
        root.setPrefSize(width,height);
        canvas.setWidth(width);
        canvas.setHeight(height);
    }
}
```

Listing 1. JavaFXDrawingFrame Constructor

Each iteration of processing a command changes the scene graph and each version is stored within a “ConcurrentLinkedDeque” [22], which is an unbounded concurrent deque based on linked nodes. This allows the JavaFX GUI thread safe access to the “JavaFXDrawingFrame” object. The deque stores up to two scene objects, in order to prevent a Nullpointer Exception in case the GUI thread accesses an empty deque. From two scenes onward, the last element of the deque is removed to keep it small. The method “setChangeSceneTrue()” changes a global boolean variable “changeFrame”, which signals the FX thread to update its GUI.

```java
// store updated scene in ConcurrentLinkedDeque for FX GUI
// store more than one scene to prevent NullPointer Exception in GUI thread
if (deque.size() < 3 && fxframe != null) {
    deque.add(fxframe);
}
else if (fxframe != null) {
    deque.removeLast();          // remove elements to keep size of deque small
    deque.add(fxframe);
```

```
}

// signal FX GUI to update
setChangeSceneTrue();
```

Listing 2: ConcurrentLinkedDeque to Store Scene Updates

While changes to the scene are being executed in the main thread, the JavaFX thread updates the GUI via an implemented “Runnable” [23] “updater” within its start method to set each new version of the scene to its stage. The updater is called every 10 milliseconds to ensure quick operations, but only refreshes if certain conditions are met, to prevent the GUI form becoming unresponsive. Each iteration checks if a scene object is within the deque, if “changeScene” signals a new available update and if an update is desired by the user (“fxWinUpdate”). When all conditions are met, the latest scene element within the deque and a new title will be set to the stage. The dimensions of the stage will be set to the scene graph’s root node in case it has been resized. Afterwards, the program evaluates changes to the stage itself, which encompass its decoration, layer, location, resizing and visibility. Lastly, the updater signals that the changes have been implemented until a new scene becomes available.

```
Thread thread = new Thread(new Runnable() {
    @Override
    public void run() {
        Runnable updater = new Runnable() {
            @Override
            public void run() {

                // check if a new scene is available and an update should be executed
                if (!deque.isEmpty() && changeScene && fxWinUpdate) {

                    // set first element of deque to stage
                    Scene scene = deque.getFirst();
                    stage.setScene(scene);
                    stage.setTitle(frameTitle);
                    stage.sizeToScene();            // in case of resize

                    // checks if changes to the stage have been signaled
                    if (changeFrame) {
                        try {
                            // change stage decoration
                            if (changeDecoration) {
                                if (stageDecorated) {
```

```
                    stage.initStyle(StageStyle.DECORATED);
                } else {
                    stage.initStyle(StageStyle.UNDECORATED);
                }
            }
            // signal that frame has been changed
            setChangeFrameFalse();

        } catch (Exception e) {
            throw new IllegalArgumentException("WinFrame "
                    + "cannot be changed once the window "
                    + " has been set to visible");
        }

        // check if stage should be set to front or back
        if (changeBackFront) {
            if (winToFront) {
                stage.toFront();
            }
            else {
                stage.toBack();
            }
        }
        // check if stage should be always on top
        if (changeAlwaysOnTop) {
            if (winAlwaysOnTop) {
                stage.setAlwaysOnTop(true);
            }
            else {
                stage.setAlwaysOnTop(false);
            }
        }

        // check if the location of the frame should be changed
        if (changeFrameLocation) {
            stage.setX(frameX);
            stage.setY(frameY);
            frameMoved();
        }

        // check if frame should be resizable
        if (fxFrameResizable) {
            stage.setResizable(true);
        } else {
```

```
                            stage.setResizable(false);
                        }

                        // check if frame should be visible
                        if (fxVisible) {
                            stage.show();
                        } else {
                            stage.hide();
                        }

                        // signal that changes to frame have been made
                        setChangeFrameFalse();
                    }

                    // signal that scene has been updated
                    setChangeSceneFalse();
                }
            }
        };
        // run updater every 10 milliseconds
        while (true) {
            try {
                Thread.sleep(10);
            } catch (InterruptedException ex) {
            }
            // UI update is run on the Application thread
            Platform.runLater(updater);
        }
    }
});
```

Listing 3. GUI Updater

The redirected input of a Rexx program, which is supplied via the “RexxRedirectingCommandHandler”, is being split into single commands and their arguments. In the method “processCommand”, the first parameter of each command (as string) is evaluated through a switch statement [24]. When the supplied command matches a case and has the correct number of arguments, the corresponding code block is executed and turns the commands into Java code. The code blocks of the cases are based on the “JavaDrawingHandler” (JDOR) but have now been modified to utilize JavaFX packages instead of awt based Java2D.

Once the last command has been processed in the main thread, the FX application thread closes automatically. Listing 4. shows an example of an original case code block from the "JavaDrawingHandler".

```
case SCALE:       // "scale x [y]" set scale for x, y; if y omitted, uses x
case SHEAR:       // "shear x [y]" set scale for x, y; if y omitted, uses x
  {
    int argNum=arrCommand.length;
    if (argNum>3)
    {
      throw new IllegalArgumentException("this command needs either no, "
          + "one or 2 arguments, received "+(arrCommand.length-1)
          +" instead");
    }

      // get current settings
    AffineTransform at=bufGC.getTransform();
    String strResult=null;
    double newX=0, newY=0;

    if (cmd==EnumCommand.SCALE)
    {
      strResult=at.getScaleX()+" "+at.getScaleY();
    }
    else // SHEAR
    {
      strResult=at.getShearX()+" "+at.getShearY();
    }

    if (argNum>1)   // set value
    {
      newX = Double.parseDouble(arrCommand[1]);
      newY = newX;    // default to X value in case Y value is omitted
      if (argNum==3)        // Y value supplied, use it
      {
        newY=Double.parseDouble(arrCommand[2]);
      }

      if (cmd==EnumCommand.SCALE)
      {
        bufGC.scale(newX,newY);
      }
      else
      {
```

```
            bufGC.shear(newX,newY);
          }
        }
        if (isOR)
        {
          if (argNum>1)
          {
            canonical=canonical+" "+newX+" "+newY;
          }
          writeOutput(slot, canonical);  // write canonical form
        }
        return strResult;       // current/old settings
      }
```

Listing 4. Case of Scale and Shear in JavaDrawingHandler

When the supplied input matches either "SCALE" or "SHEAR", the program evaluates if the correct number of arguments are supplied. If this condition is false, an error message will be returned, otherwise a new "AffineTransform" [25] object "at" is created and set as the current transform state of the "Graphics2D" object "bufGC". The Graphics2D [10] class offers capabilities for managing geometry, coordinate transformations, color, and text arrangement, which are applied to its corresponding "BufferedImage". The "AffineTransform" class enables 2D affine transformations, linear mappings between 2D coordinates while preserving the geometric properties of lines, achieved through sequences of translations, scales, flips, rotations, and shears.

Next, a new string variable "strResult" is created and stores the current state of either scale or shear properties of "at", depending on the received command. If more than one argument has been supplied (aside from the command itself), then it is parsed and stored in a newly created double variable "newX". The double variable "newY" is either set to match this value, or, if two arguments are supplied, parses the second argument. Depending on the command, either new scale or new shear parameters are added to the Graphics2D object "bufGC" with the values of "newX" and "newY". Lastly, the changes can optionally be stored in the variable "cannonical", which supplies the programmer with information on the made changes, and "strResult" is returned.

Listing 5. shows the modified version of the same switch statements that are implemented in the "JavaFXDrawingHandler". However, instead of "AffineTransform" [25] it implements its JavaFX counterpart Affine [26], which uses similar transformations but also allows transformations in 3D space. In the provided case, a newly created Affine object "fxAffine" is set to the current state of the "GraphicsContext" [27] "canGC", which modifies the graphical elements of its “Canvas” [28]. The command arguments are then again parsed and stored in double variables. The new transformation is then applied to “canGC”.

However, unlike in the “JavaDrawingHandler”, where “Graphics2D” allows to add instances of awt based Java2D shape classes to the image, the “GraphicsContext” in the “JavaFXDrawingHandler” cannot add objects of JavaFX’ shape class [29] to its canvas. Such shapes need to be added to the node “shapeGroup” to be displayed on the scene graph. Conversely, transformations such as the previously mentioned “scale” and “shear” can only be applied by “canGC” to drawing elements that are directly placed on the canvas. Hence, the currently stored transformation must be parsed from “GraphicsContext” and applied to a shape before setting it to the “shapeGroup”.

```
case SCALE:       // "scale x [y]" set scale for x, y; if y omitted, uses x
case SHEAR:       // "shear x [y]" set scale for x, y; if y omitted, uses x
    {
        int argNum=arrCommand.length;
        if (argNum>3)
        {
            throw new IllegalArgumentException("this command needs either no, "
                    + "one or 2 arguments, received "+(arrCommand.length-1)
                    +" instead");
        }

        // get current settings
        Affine fxAffine = canGC.getTransform();
        String strResult=null;
        double newX=0, newY=0;

        if (cmd==EnumCommand.SCALE)
        {
            strResult=fxAffine.getMxx()+" "+fxAffine.getMyy();
        }
        else // SHEAR
        {
            strResult=fxAffine.getMxy()+" "+fxAffine.getMyx();
        }

        if (argNum>1)   // set value
        {
            newX = Double.parseDouble(arrCommand[1]);
            newY = newX;    // default to X value in case Y value is omitted
```

```java
        if (argNum==3)        // Y value supplied, use it
        {
          newY=Double.parseDouble(arrCommand[2]);
        }

        if (cmd==EnumCommand.SCALE)
        {
          // set new Scale values
          fxAffine.appendScale(newX,newY);
          // apply affine transform to canvas
          canGC.setTransform(fxAffine);
        }
        else
        {
          // set new Shear values
          fxAffine.appendShear(newX,newY);
          // apply affine transform to canvas
          canGC.setTransform(fxAffine);
        }
      }
      if (isOR)
      {
        if (argNum>1)
        {
          canonical=canonical+" "+newX+" "+newY;
        }
        writeOutput(slot, canonical);  // write canonical form
      }
      return strResult;       // current/old settings
    }
```

Listing 5. Case of Scale and Shear in JavaFXDrawingHandler

4.2. Packages in Java

After "JavaFXDrawingHandler" is completed, it can now be turned into a package [30]. Within the file, the command "package org.oorexx.handlers.jdorfx" will be added on top of the program. Its name will become the file path. A Java package can be created by opening a command terminal in the folder where "JavaFXDrawingHandler" is saved and entering the following commands:

1. C:\Users*Your Name*>javac JavaFXDrawingHandlere.java

   Save and compile the file

2. C:\Users*Your Name*>javac -d . JavaFXDrawingHandlere.java

   Compile Package

```
PS C:\Users\schal\Desktop> javac .\JavaFXDrawingHandler.java
PS C:\Users\schal\Desktop> javac -d . JavaFXDrawingHandler.java
PS C:\Users\schal\Desktop>
```

Fig. 12. Java Package

These commands create a new folder "\org\oorexx\handlers\jdorfx" which contains the compiled classes. Next, the folder will be turned into a "JAR" file named "JDORFX_20240505.jar" with the following command:

3. C:\Users\schal\Desktop> jar -cvf JDORFX_20240505.jar org

```
PS C:\Users\schal\Desktop> jar -cvf JDORFX_20240505.jar org
Manifest wurde hinzugefügt
org/ wird hinzugefügt(ein = 0) (aus = 0)(0 % gespeichert)
org/oorexx/ wird hinzugefügt(ein = 0) (aus = 0)(0 % gespeichert)
org/oorexx/handlers/ wird hinzugefügt(ein = 0) (aus = 0)(0 % gespeichert)
org/oorexx/handlers/jdorfx/ wird hinzugefügt(ein = 0) (aus = 0)(0 % gespeichert)
org/oorexx/handlers/jdorfx/EnumCommand.class wird hinzugefügt(ein = 12894) (aus = 6768)(47 % verkleinert)
org/oorexx/handlers/jdorfx/EnumShape.class wird hinzugefügt(ein = 4378) (aus = 2217)(49 % verkleinert)
org/oorexx/handlers/jdorfx/JavaFXDrawingFrame.class wird hinzugefügt(ein = 1610) (aus = 917)(43 % verkleinert)
org/oorexx/handlers/jdorfx/JavaFXDrawingHandler$1.class wird hinzugefügt(ein = 987) (aus = 564)(42 % verkleinert)
org/oorexx/handlers/jdorfx/JavaFXDrawingHandler$2$1.class wird hinzugefügt(ein = 2509) (aus = 1405)(44 % verkleinert)
org/oorexx/handlers/jdorfx/JavaFXDrawingHandler$2.class wird hinzugefügt(ein = 1076) (aus = 582)(45 % verkleinert)
org/oorexx/handlers/jdorfx/JavaFXDrawingHandler$3.class wird hinzugefügt(ein = 6073) (aus = 2597)(57 % verkleinert)
org/oorexx/handlers/jdorfx/JavaFXDrawingHandler$ConditionType.class wird hinzugefügt(ein = 1204) (aus = 549)(54 % verkle
inert)
org/oorexx/handlers/jdorfx/JavaFXDrawingHandler.class wird hinzugefügt(ein = 80765) (aus = 35176)(56 % verkleinert)
PS C:\Users\schal\Desktop>
```

Fig. 13. JAR file

"JAR", short for "Java Archive", is a file format based on the "ZIP" format, designed for lossless data compression, archiving, decompression, and archive unpacking [31].

4.3. JDORFX

After the Java package has been successfully built, a new ooRexx cls file can be created, which allows other Rexx programs to access the "JavaFXDrawingHandler" when being addressed.

```
::routine addJdorFXHandler   public
  use strict arg environmentName="JDORFX"

  call BsfCommandHandler "add", -
  environmentName, -
  .bsf~new("org.oorexx.handlers.jdorfx.JavaFXDrawingHandler")

::requires "BSF.CLS"    -- get ooRexx-Java bridge
```

Listing 6. jdorfx.cls

JDORFX serves as a new command handler that supplies the environment through the optional “environmentName”, which will be set to “JDORFX” if no argument is supplied. It utilizes the previously created “org.oorexx.handlers.jdorfx.JavaFXDrawingHandler” package. Lastly, it requires the “BSF.CLS” file, which enables ooRexx to use Java functionalities.

4.4. Environment for Nutshell Examples

The last stage of development requires setting the environment, which will enable the execution of Rexx programs that integrate JDORFX. First, when writing a program, the following block of code needs to be implemented:

```
-- create JDORFX handler
-- load and add the Java Rexx command handler, using default name: JDORFX
call addJdorFXHandler
-- set default environment to JDORFX
address jdorfx
…

-- get ooRexx-Java bridge, contains JDORFX Rexx command handler
::requires "jdorfx.CLS"
```

Listing 7. Code to Address JDORFX in jdorfx_shapes2d.rxj

Rexx recognizes three types of instructions: assignment, keyword, e.g. “do” or “call”, and command [32]. Command instructions are sent to the operating system by default. However, using the keyword “address” allows to specify a different Rexx command handler instead, such as JDORFX in the example above. ooRexx further adds directive instructions to Rexx. During the execution of a Rexx program, it undergoes three phases. Firstly, the syntax is checked in the "Load" phase, followed by the execution of directives in the "Setup" phase. Finally, the program proceeds to execute the first instruction at the beginning of the program in the "Execution" phase. This allows directives to be used to set up and configure the execution environment, which is achieved through "::requires 'jdorfx.cls'" in the given example.

In order to ensure that the Rexx program can access all necessary resources, "jdorfx.cls" and "JDORFX_20240505.jar" must be saved in specific directories. Initially, Rexx programs look for required classes within the current folder. If the classes are not found there, the program will then search in the "BSF4ooRexx850" directory. Hence, "jdorfx.cls" should either be placed in the same directory as the Rexx program that requires it or in "%WINPROFILE%\BSF4ooRexx850". Secondly, the file "JDORFX_20240505.jar" should be placed in "%WINPROFILE%\BSF4ooRexx850\lib", which has been designated as the "CLASSPATH" environment variable for Java classes during the installation of BSF4ooRexx850 [33].

5. JDORFX Commands and Examples

After the completion of the “JavaFXDrawingHandler”, its graphic capabilities are being tested. The following nutshell examples compare the 2D drawing capabilities of JDORFX and JDOR. Two Rexx programs containing the same commands are executed, one uses JDOR while the other uses JDORFX. The objective is to run both programs and compare the output on each created window.

After the comparison, a list of commands is provided that explains 3D functionalities which are only accessible through JDORFX and showcase examples of their implementation are provided.

5.1. Comparing JDOR and JDORFX - Drawing

Drawing with JDORFX works similar to drawing with JDOR. While the “JavaDrawingHandler” uses “Graphics2D” [10] class to draw onto a “BufferedImage”, the “JavaFXDrawingHandler” utilizes the “GraphicsContext” [27] class to draw onto a canvas which is embedded in the scene graph.

The provided nutshell examples in Fig. 11 shows the output of Rexx programs which implement either JDOR or JDORFX. They illustrate drawing operations which are executed on their respective GUI. In both cases, the position of shapes and their appearance are identical. However, some differences in color tones can be observed. This is due to the fact that awt based Java2D “colors” and JavaFX “colors” use different color codes e.g. for green. The first uses the RGB value of 0-204-0 [34], the latter uses the HEX value #008000, which translates to the RGB value of 0-128-0 [35].

```
-- create JDORFX handler
-- load and add the Java Rexx command handler, using default name: JDORFX
call addJdorFXHandler
-- set default environment to JDORFX
address jdorfx

say "Create a new scene and draw on canvas"
-- creating a new window with size 500 x 500
```

```
newimage 500 500
-- draw filled polygon with nPoints taken from xPoints-array and yPoints-array
fillpolygon "(10,30,40,50,110,140)" "(0,100,40,50,200,0)" 6
-- move current x and y points
moveto 200 50
-- draw filled arc with width heigth start angle and arc angle
fillarc 100 200 40 120
-- move current x and y points
moveto 400 50
-- set draw and fill color to red
color red
-- draw line from current x and y points to new x and y
drawline 400 200
-- move current x and y points
moveto 50 300
-- draw filled oval with width and height
filloval 50 100
-- set draw and fill color to green
color green
-- move current x and y points
moveto 200 300
-- draw filled rectangle with width and height
fillrect 50 100
-- move current x and y points
moveto 300 300
-- draw round rectangle with width heigth arcwidth and archheight
drawroundrect 100 150 10 10
-- show created window
winshow
say
say "Sleep for 5 seconds and end program..."
sleep 5

-- get ooRexx-Java bridge, contains JDORFX Rexx command handler
::requires "jdorfx.CLS"
```

Listing 8. jdorfx_drawing2d.rxj

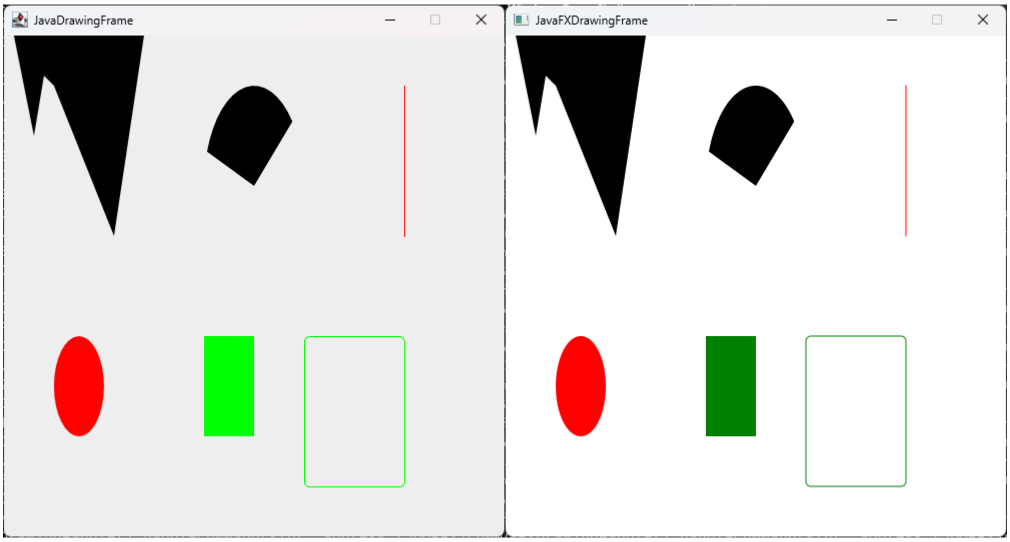

Fig. 14. JDOR vs JDORFX - drawing2d.rxj

5.2. Comparing JDOR and JDORFX - 2D Shapes

While the “Graphics2D” class implemented in the “JavaDrawingHandler” allows to draw awt based Java2D shape objects onto the “BufferedImage”, the “GraphicsContext” of JavaFX cannot add shape [29] objects to the corresponding canvas.

Hence, instances of Shape subclasses, e.g. circle, line, rectangle…, are created and stored in a hashmap [36]. When the commands “*drawshape*” or “*fillshape*” are used, the shape instances are added to the group [21] “shapeGroup” of the scene graph instead of the canvas. Since the “shapeGroup” itself cannot store or set color properties of its children, the currently set color of the “GraphicsContext” is parsed and applied to a shape when it is drawn. This also relates to stroke properties.

Fig. 12. shows the outputs of JDORFX and JDOR while drawing shapes with various stroke and color properties. While their mode of operations behind the scenes differ, their result remains the same, aside from the differences in color tones.

```
-- create JDORFX handler
-- load and add the Java Rexx command handler, using default name: JDORFX
call addJdorFXHandler
-- set default environment to JDORFX
address jdorfx

say "Create a scene and add 2D shapes"
-- creating a new window with size 500 x 500
newimage 500 500
-- set draw and fill color to blue
```

```
color blue
-- set stroke with width=3 cap=2 join=2 miterLimit=10 dashArray=(10,20) dashOffset=4
stroke myStroke 3 2 2 10 "(10,20)" 4
stroke myStroke
-- draw filled polygon with nPoints taken from xPoints-array and yPoints-array
shape myPoly polygon "(50,150,100)" "(50,50,100)" 3
fillshape myPoly
-- draw arc with x y width heigth start extend type
shape myArc arc 200 50 100 100 0 250 2
drawshape myArc
-- set draw and fill color to R=0.5 G=0.3 B=0.6 Alpha=0.9
color myColor 0.5 0.3 0.6 0.9
color myColor
-- draw line from x  y to newX and newY
shape myLine line 350 50 300 200
drawshape myLine
-- draw ellipse with x y width height
shape myEllipse ellipse 400 50 50 100
drawshape myEllipse
-- set draw and fill color to orange
color orange
-- draw rectangle with x y width height
shape rec rectangle2d  50 300 100 100
drawshape rec
-- draw filled round rectangle with x y width heigth arcWidth archHeight
shape myRoundrect roundrect 200 300 50 100 20 20
fillshape myRoundrect
-- set draw and fill color to gray
color gray
-- draw cubic cruve with x1 y1 ctrlx1 ctrly1 ctrlx2 ctrly2 x2 y2
shape myCubic cubic 300 300 300 350 400 350 300 400
drawshape myCubic
-- draw cubic cruve with x1 y1 ctrlx ctrly x2 y2
shape myQuadCurve quadcurve 450 450 300 300 400 300
drawshape myQuadCurve
-- set draw and fill color to pink
color pink
-- create a path and add elements
shape myPath path 1
pathmoveto myPath 50 250
-- add curve to path with ctrlx1 ctrly1 ctrlx2 ctrly2 x y
pathcurveto myPath 100 150 200 300 200 250
pathlineto myPath 400 250
--draw path
```

```
drawshape myPath
-- show created window
winshow
say
say "Sleep for 5 seconds and end program..."
sleep 5

-- get ooRexx-Java bridge, contains JDORFX Rexx command handler
::requires "jdorfx.CLS"
```

Listing 9. jdorfx_shapes2d.rxj

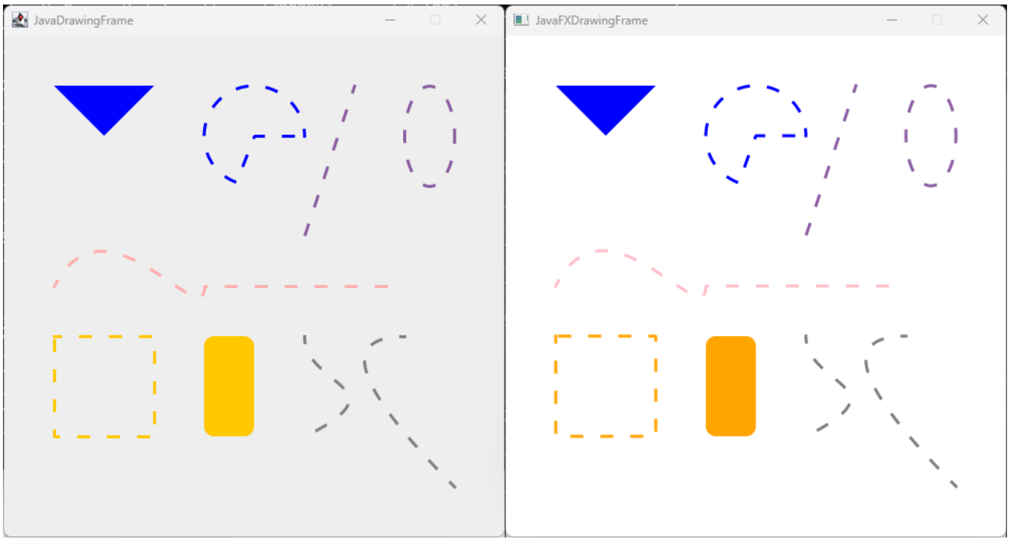

Fig. 15. JDOR vs JDORFX - shapes2d.rxj

5.3. Comparing JDOR and JDORFX - 2D Transformations

In "JavaDrawingHandler", "AffineTransformations" [25] states are stored in the Graphics2D class and applied to its elements when they are drawn. Similarly, in "JavaFXDrawingHandler" "Affine" transformations [26] are stored in the "GraphicsContext" class and applied to its canvas. However, since the shapes added to the "shapeGroup" are not part of the canvas, the previously defined "affine transformations need to be parsed form the "GraphicsContext" and set to the shapes when they are drawn.

The following example shows the use of transformation commands. Transformations can either be set to all drawn elements or applied to a single path.

```
-- create JDORFX handler
-- load and add the Java Rexx command handler, using default name: JDORFX
call addJdorFXHandler
-- set default environment to JDORFX
address jdorfx

say "Create a scene and add and transform 2D shapes"
-- creating a new window with size 500 x 500
newimage 500 500
-- transform following shapes and drawing elements with
-- translateX translateY scaleX scaleY shearX shearY
transform  50 50 1.1 1.1 0.2 0.2
-- move current x and y points
moveto 0 0
-- draw line from current x and y points to new x and y
drawline 0 200
-- rotate following shapes and drawing elements by angle
rotate (-20)
-- draw ellipse with x y width height
shape myEllipse ellipse 25 50 50 100
drawshape myEllipse
-- scale following shapes and drawing elements by y=1.5
scale 1 1.5
-- draw rectangle with x y width height
shape myRect rectangle2d  100 50 50 100
drawshape myRect
-- shear following shapes and drawing elements by x=0.2 y=0.6
shear 0.2 0.6
-- draw arc with x y width height start length type
shape myArc arc 175 (-50) 100 100 0 120 2
drawshape myArc
-- set draw and fill color to red
color red
-- create a path myPath and add elements
shape myPath path 1
pathmoveto myPath 300 50
pathlineto myPath 400 50
pathlineto myPath 400 100
pathlineto myPath 300 100
pathclose mypath
-- create transformation with name myTransform
transform myTransform (-350) 50 1 1.5 0.5 0
-- use transformation myTransform on path myPath
pathtransform myPath myTransform
```

```
--draw path
fillshape myPath
-- show created window
winshow
say
say "Sleep for 5 seconds and end program..."
sleep 5

-- get ooRexx-Java bridge, contains JDORFX Rexx command handler
::requires "jdorfx.CLS"
```

Listing 10. jdorfx_transform2d.rxj

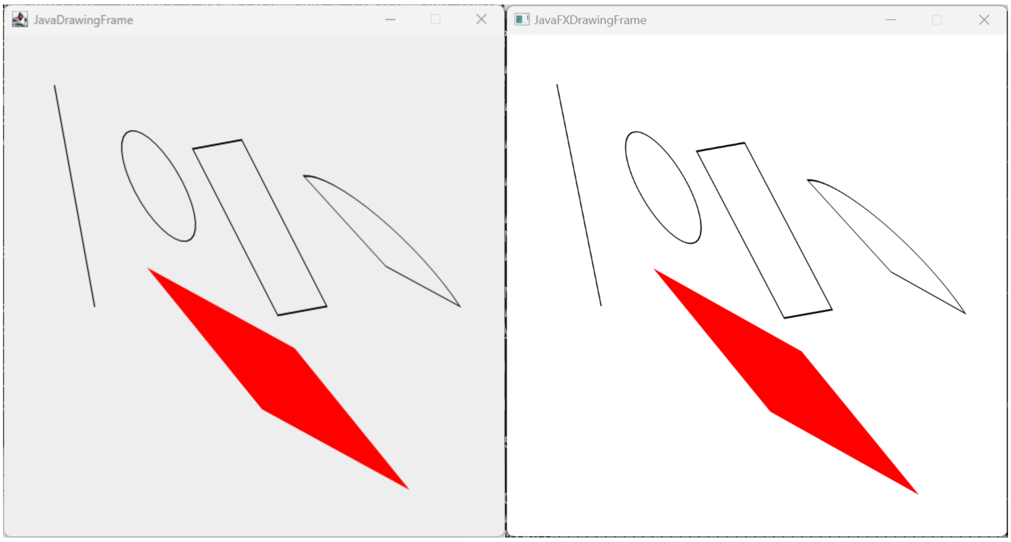

Fig. 16. JDOR vs JDORFX - transform2d.rxj

5.4. Command List

After comparing the output of 2D capabilities of JDOR and JDORFX, a list of commands is provided that correspond to 3D functionalities which are only accessible through JDORFX. Afterwards, nutshell examples are provided which describe the commands in more detail.

Command	Arguments	Description
shape3D	String *nickname*	Queries the 3D shape with the supplied nickname. If no such shape exists, return error message.

	String *nickname* *“box”* Double *x y z* Double *width height depth*	Creates a new named box at position x,y,z, a size of width, height, depth and stores it.
	String *nickname* *„cylinder“* Double *x y z* Double *radius height*	Creates a new named cylinder at position x,y,z, a size of radius and height and stores it.
	String *nickname* *„sphere“* Double *x y z* Double *radius*	Creates a new named sphere at position x,y,z, a radius and stores it.
drawShape3D *or: draw3DShape*	String *nickname*	Draws the supplied 3D shape as wire frame model. If no such shape exists, return error message.
fillShape3D *or: fill3DShape*	String *nickname*	Draws the supplied 3D shape with filled vertices. If no such shape exists, return error message.
camera		If no arguments supplied, query the last camera that has been set to the scene.
	String *nickname*	Queries the camera with the supplied nickname. If no such camera exists, return error message.
	String *nickname* *“parallel”* Double x y z	Creates a new named parallel camera at position x,y. The z value does not influence its location. A parallel camera looks at the xy plane and possess a viewing volume for parallel projection.

	String *nickname* *"perspective"* Double x y z [Double *fieldOfView* (optional)]	Creates a new named perspective camera at position x,y,z and the optional argument fieldOfView. If the argument fieldOfView is not supplied, it is set to the default of 30. A perspective camera looks at the xy plane and defines the viewing volume for a perspective projection (fieldOfView).
setCamera		If no arguments are supplied, sets the scene camera back to its default parallel camera at position 0 0 0.
	String *nickname*	Sets the named camera as the new scene camera. If no such camera exists, return error message.
light	String *nickname*	Queries the light with the supplied name. If no such light node exists, return error message.
	String *nickname* *"ambient"* [String *color* (optional)]	Creates a new named ambient light with an optional color value. If no color value is supplied, the default color is white. An ambient light is a light source that radiates from all directions.
	String *nickname* *"point"* Double *x y z* [String *color* (optional)]	Creates a new named point light at position x, y, z and with an optional color value. If no color value is supplied, the default color is white. A point light projects light in all directions away from its position.
setLight	String *Nickname* [String *turnOn/turnoff* (optional)]	Sets the named light source into the scene. If the optional argument is set to *turnOn* or the argument is not supplied, the light source will be turned on. If the optional argument is set to *turnOff*, the named light source will be turned off.

		If no such light exists, return error message.
rotateShape3D *or: rotate3DShape*	String *nickname* Double *angle* Double *pivotX pivotY pivotZ* Double *axisX axisY axisZ*	Rotates the named 3D shape at an angle with the coordinates for its pivot point around the set axis. The axis values define the coordinate magnitude of the rotation axis (0 – 1.0). Values of 0 0 1 rotates the shape on the z axis. If no such 3D shape exists, return error message.
scaleShape3D *or: scale3DShape*	String *nickname* Double *x y z*	Scales the named 3D shape along the set axis. Values of 2 0 0 doubles the size of the shape along the x axis. If no such 3D shape exists, return error message.
shearShape3D *or: shear3DShape*	String *nickname* Double *x y*	Shears the named 3D shape along the set axis. If no such 3D shape exists, return error message.
translateShape3D *or: translate3DShape*	String *nickname* Double *x y z*	Moves the named 3D shape along the set axis. If no such 3D shape exists, return error message.
map	String *NickName* String *imagePath*	Stores an image with given file path as named variable.
	String *NickName* String *imagePath* Integer *addWidth* Integer *addHeight* Double *angle* [String *color* (optional)]	Creates a new named image based on the provided image with file path. addWidth and addHeight will be added to the image's size and will change its proportions. If the values are negative, the image will be cropped. The image will be rotated at the provided angle clockwise around its centre.

		A rotation value beyond a magnitude of 90 degrees may lead to loss of image quality. If an optional color value is supplied, all transparent pixels of the image will be filled with said color.
material	String *Nickname* [String *color* (optional)]	Creates a new named material with an optional color value. If no argument is provided, then the default color is white.
materialColor	String *NickName* String *colorType* String *color* [Double *specularPower* (optional)]	Targets the named material and sets the color property of the chosen colorType (*diffuse/diffuseColor* or *specular/ specularColor*). If the chosen colorType is specular, then the specularPower can be set. If no argument is provided, its default value is 32. If no such material exists, return error message. A diffuse color represents the base color of a surface material. A specular color is the color value of light that is reflecting from a surface material. If no specular color is set, the default specular color is white. The specular power indicates the level of smoothness of the material. The smaller the value, the narrower the reflections and the smoother the surface appears.
materialMap	String *NickName* String *mapType* String *imagePath/nickname*	Adds a new image as the chosen mapType (*bump/bumpMap, diffuse/diffuseMap, selfIllumination/selfIlluminationMap or specular/ specularMap*) to a named material. The image is selected via its file path or a previously saved nickname as string. If no such material or image exist, return error message. A bump map acts as a normal map as RGB image for a material. It adds depth to the material's surface image.

		A diffuse map uses an image as the surface of a material. A self-illumination map lets light emanate from a material. A specular map defines the reflection properties of a material.
setMaterial	String *sNickname* String *mNickname*	Sets the named material to the named 3D shape. If no such 3D shape or material exist, return error message.

Tab. 1: List of 3D Commands in JDORFX

5.5. Drawing 3D Shapes

The first example shows the implementation of JavaFX' "box" [37], "cylinder" [38] and "sphere" [39], which are all subclasses of "Shape3D" [40], which stores common properties for 3D geometric shapes, such as material [41], "drawMode" [42] and "cullFace" [43]. Similar to 2D shapes, the created 3D shapes are added to the group node "shape3DGroup", which is embedded in the scene graph.

The command "*shape3d*" instantiates such a shape3D object and uses the following arguments for all subclasses:

- "String *nickName*": set a name for the newly created shape.
- "String *shapeType*": choose the type of 3D shape (box, cylinder or sphere).
- "Double *x y z*": the coordinates of the 3D shape within the scene. In contrast to 2D shapes, its anchor point is the shape's center, which is set to the upper left corner of the scene when the coordinates are x=0 and y=0).

The following arguments depend on the previously chosen type of 3D shape to define its size.

- Box: "Double *width height depth*"
- Cylinder: "Double *radius height*"
- Sphere: "Double *radius*"

After its creation, the 3D shape gets stored and can be added to the scene graph with the commands "*draw3DShape*" or "*fill3DShape*". For easier access, the synonyms "*drawShape3D*" and "*fillShape3D*" respectively have been implemented. Both commands only use one argument, which is the "*nickName*" of the 3D shape one wishes to add. Using "*draw3DShape*", sets a 3D shapes "drawMode" property [42] to "DrawMode.LINE" and draws the 3D object as a see through wireframe model. "*fill3DShape*" on the other hand sets its "drawMode" property to "DrawMode.FILL", which fills its interior.

The following example shows the implementation of all three basic 3D shapes. The shapes still appear 2 dimensional because no perspective camera has been set to the scene, which will be addressed in the next example.

```
-- create JDORFX handler
-- load and add the Java Rexx command handler, using default name: JDORFX
call addJdorFXHandler
-- set default environment to JDORFX
address jdorfx

say "Create a scene and add 3D shapes"
-- creating a new window with size 500 x 500
newimage 500 500
-- create new sphere with the name "mySphere" at location x=100, y=100, z=0
-- and a radius=50
shape3d mySphere sphere 100 100 0 50
-- create new box with the name "myBox" at location x=400, y=175, z=0
-- and size: width=100, height=250, depth=200
shape3d myBox box 400 175 0 100 250 200
-- create new cylinder with the name "myCylinder" at location x=100, y=350, z=0
-- and size: radius=50, height=150
shape3d myCylinder cylinder 100 350 0 50 150
-- place mySphere onto scene with filled vertices; also: fillshape3d
fill3dshape mySphere
-- place myBox onto scene as line / wireframe model; also: drawshape3d
draw3dshape myBox
-- place myCylinder onto scene with filled vertices; also: fill3dshape
fillshape3d myCylinder
-- show created window
winshow
say
say "Sleep for 5 seconds and end program..."
sleep 5

-- get ooRexx-Java bridge, contains JDORFX Rexx command handler
::requires "jdorfx.CLS"
```

Listing 11. jdorfx_shapes3d.rxj

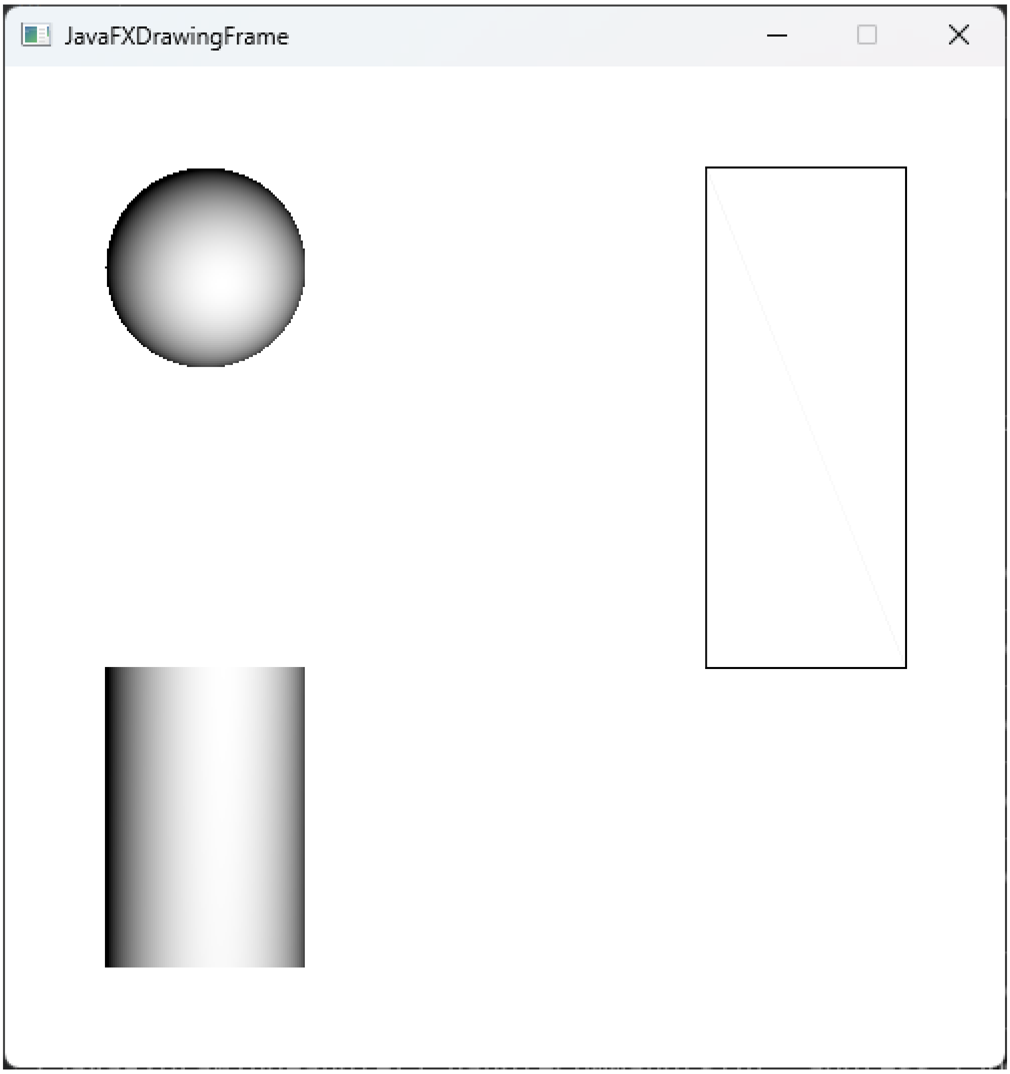

Fig. 17. Output jdorfx_shapes3d.rxj

5.6. Parallel and Perspective Camera

The “Camera” class [44] defines how a scene’s coordinate space is mapped onto the window. It has two subclasses, “ParallelCamera” [45] and “PerspectiveCamera” [46]. The default camera of a scene is a “ParallelCamera” object with its position set at the centre of the scene, looking at the xy plane. It possesses a viewing volume for parallel projection and it cannot be moved along the z axis. On the other hand, a “PerspectiveCamera” defines the viewing volume for a perspective projection, which can be changed with the value of its “fieldOfView” property. It can be moved along the z axis.

The command “*camera*” creates a new camera object and uses the following parameters for both subclasses:

- “String *nickName*”: set a name for the newly created camera.
- “String *cameraType*”: choose the type of camera (*perspective or parallel*).

- "Double *x y z*": the coordinates of the camera. Setting the coordinates to 0 0 0 places the camera in the centre of the scene. A "parallelCamera" cannot be moved along the z axis, hence changing its z does nothing.

A "PerspectiveCamera" has one additional optional argument:

- "Double *fieldOfView*" [optional]: sets the field of view angle of the camera's projection plane. If this argument is not supplied, the value will be set to the default of 30.

The command "*setCamera*" with the optional argument "*nickName*" sets the named object as the scene camera. If no "*nickName*" is supplied, the scene camera will be set back to its original default "ParallelCamera".

The following example shows a scene which switches between "PerspectiveCameras" and "ParallelCameras" during the runtime of its program. The cameras are set at different positions, while the shown 3D shapes remain stationary. In Listing 12. the front face of the box shape seems to disappear. This happens because the box is drawn as a wireframe model and the properties of the scene's only "LightBase" is set as ambient and white. The "LightBase" class will be discussed in the next example.

```
-- create JDORFX handler
-- load and add the Java Rexx command handler, using default name: JDORFX
call addJdorFXHandler
-- set default environment to JDORFX
address jdorfx

say "Create a scene with the default parallel camera"
-- creating a new window with size 500 x 500
newimage 500 500
-- show created window
winshow
-- create new sphere with name "mySphere" at location x=50, y=200, z=0 and a radius=50
shape3d mySphere sphere 205 225 0 50
-- create new box with name "myBox" at location x=200, y=200, z=0
-- and size: width=100, height=100, depth=100
shape3d myBox box 325 150 0 100 100 100
-- create new cylinder with name "myCylinder" at location x=350, y=200, z=0
-- and size: radius=50, height=200
shape3d myCylinder cylinder 325 350 0 50 100
-- place mySphere onto scene with filled vertices; also: fillshape3d
fill3dshape mySphere
-- place myBox onto scene as line / wireframe model; also: drawshape3d
```

```
draw3dshape myBox
-- place myCylinder onto scene with filled vertices; also: fillshape3d
fill3dshape myCylinder
say
say "Sleep for 5 seconds and then create a new perspective camera"
say "Location: x=-50 y=-100 z=-100, default fieldOfView=30"
sleep 5
--create perspectiveCamera1
camera perspectiveCamera1 perspective (-50) (-100) (-100)
-- set view to perspectiveCamera1
setcamera  perspectiveCamera1
say
say "Sleep for 5 seconds and then create a new perspective camera"
say "Location: x=200 y=0 z=-50, fieldOfView=150"
sleep 5
--create perspectiveCamera2
camera perspectiveCamera2 perspective 200 0 (-50) 150
-- set view to perspectiveCamera1
setcamera  perspectiveCamera2
say
say "Sleep for 5 seconds and then create a new parallel camera"
say "Location: x=0 y=100 z=-200"
say "Note: The z variable has no impact on a parallel camera"
sleep 5
--create parallelCamera
camera parallelCamera parallel 0 100 (-200)
-- set view to parallelCamera
setcamera  parallelCamera
say
say "Sleep for 5 seconds and end program..."
sleep 5

 -- get ooRexx-Java bridge, contains JDORFX Rexx command handler
::requires "jdorfx.CLS"
```

Listing 12. jdorfx_camera.rxj

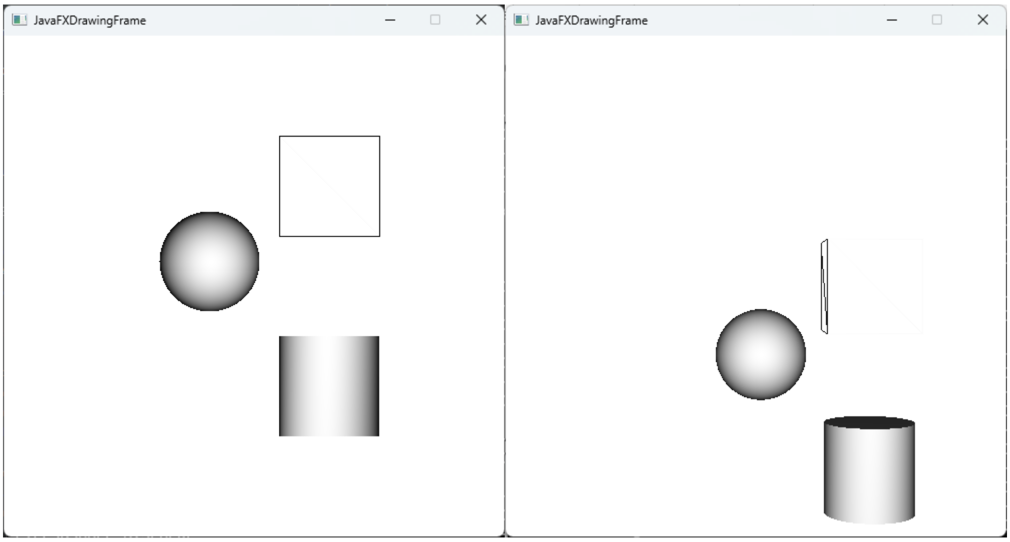

Fig 18. Parallel Camera vs Perspective Camera 1 jdorfx_camera.rxj

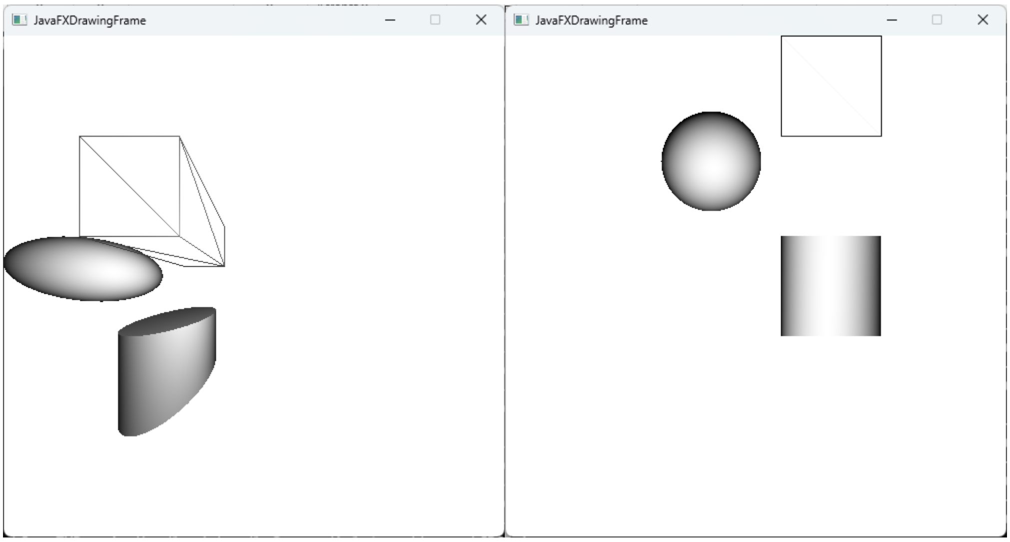

Fig. 19. Perspective Camera vs Parallel Camera 2 jdorfx_camera.rxj

5.7. Light

In order to change the lighting of a scene, a new “LightBase” [47] object can be added, which has properties of “color” and “lightOn”. “LightBase” has two subclasses, “AmbientLight” [48] and “PointLight” [49]. The default lighting of a scene is white “AmbientLight”, which is a light source radiating from all directions. In contrast, a “PointLight” can be placed anywhere in 3D space and projects light in all directions away from its position.

The implemented command “*light*” instantiates a new “LightBase” object and has the following arguments for both subclasses:

- “String *nickName*”: set a name for the newly created light.

- "String *lightBaseType*": choose the type of "lightBase" (*ambient or point*)
- "String *color*" [optional]: optional argument to set color of the light. If no such argument is supplied, the default color will be white.

The class "*PointLight*" additionally requires arguments for its position:

- "Double *x y z*": the coordinates of the "PointLight". Setting the coordinates to 0 0 0 places the light at the upper left corner of the scene.

The new light will be added to the scene automatically after its creation, but it will be turned off. The command "*setLight*" has the following arguments:

- "String *nickName*": targets the camera with this name.
- "String *turnOn* / *turnOff*" [optional]: if the parameter is "*turnOff*", the named light will be turned off. If the parameter is "*turnOn*" or not supplied at all, the light will be turned on.

The following example shows a combination of different light sources at different times in the same program. Fig. 17 shows the scene once with only a red "AmbientLight" and once with two additional "PointLights". Using only an "AmbientLight" can make it difficult to perceive the spatiality of 3D shapes. In Fig. 18 the "AmbientLight" is turned off again.

```
-- create JDORFX handler
-- load and add the Java Rexx command handler, using default name: JDORFX
call addJdorFXHandler
-- set default environment to JDORFX
address jdorfx

say "Create a new scene and set ambient light to red"
-- creating a new window with size 500 x 500
newimage 500 500
-- show created window
winshow
-- create new box shapes
shape3d ceiling box 250 110 0 500 20 500
shape3d floor box 250 410 0 500 20 500
-- place box shapes onto scene
fill3dshape ceiling
fill3dshape floor
-- create new cylinder shapes
shape3d leftPillar1 cylinder 10 260 (-200) 10 280
shape3d leftPillar2 cylinder 10 260 (-100) 10 280
shape3d leftPillar3 cylinder 10 260 0 10 280
shape3d leftPillar4 cylinder 10 260 100 10 280
```

```
shape3d leftPillar5 cylinder 10 260 200 10 280
shape3d rightPillar1 cylinder 490 260 (-200) 10 280
shape3d rightPillar2 cylinder 490 260 (-100) 10 280
shape3d rightPillar3 cylinder 490 260 0 10 280
shape3d rightPillar4 cylinder 490 260 100 10 280
shape3d rightPillar5 cylinder 490 260 200 10 280
-- place cynlinder shapes onto scene
fill3dshape leftPillar1
fill3dshape leftPillar2
fill3dshape leftPillar3
fill3dshape leftPillar4
fill3dshape leftPillar5
fill3dshape rightPillar1
fill3dshape rightPillar2
fill3dshape rightPillar3
fill3dshape rightPillar4
fill3dshape rightPillar5
-- create and add new sphere to scene
shape3d mySphere sphere 250 250 200 50
draw3dShape mySphere
-- create new perspective camera and set scene camera
camera perspectiveCamera perspective 0 0 (-150)
setcamera  perspectiveCamera
-- create and add new ambient light to scene
light ambientLight ambient red
-- turn ambientLight on
setlight ambientLight
-- create and add new point lights to scene
light blueLight point 400 200 (-100) blue
light greenLight point 100 200 100 green
say
say "Sleep for 5 seconds and turn on point lights"
say "Location point light blue: x=400 y=200 z=-100"
say "Location point light green: x=100 y=200 z=100"
sleep 5
-- turn point lights on
setlight blueLight
setlight greenLight
say
say "Sleep for 5 seconds and turn off ambient light"
sleep 5
-- turn ambientLight off
setlight ambientLight turnoff
say
```

```
say "Sleep for 5 seconds and end program..."
sleep 5

-- get ooRexx-Java bridge, contains JDORFX Rexx command handler
::requires "jdorfx.CLS"
```

Listing 13. jdorfx_light.rxj

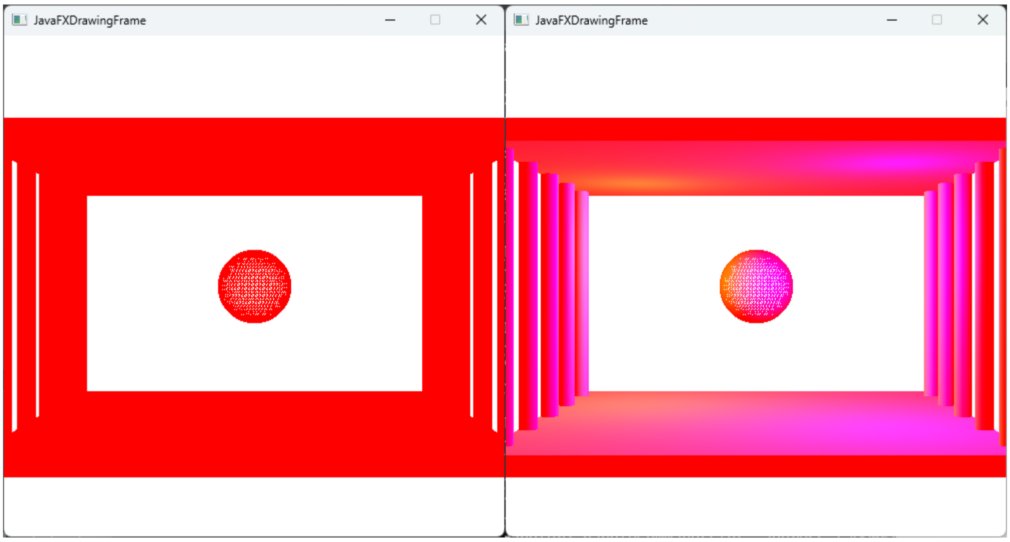

Fig. 20.AmbientLight vs PointLight jdorfx_light.rxj

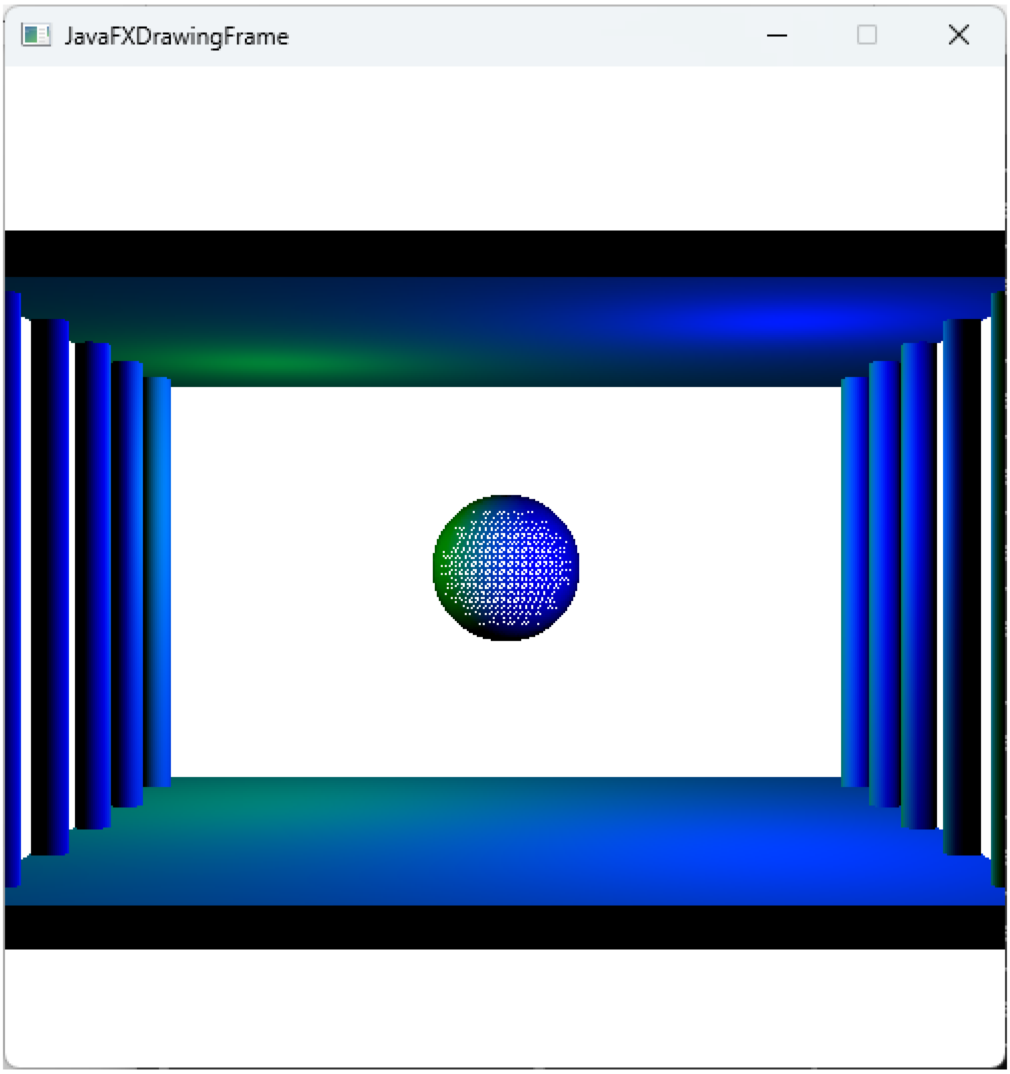

Fig. 21. PointLight jdorfx_light.rxj

5.8. Transform 3D Shapes

Transformations of 3D shapes are similar to those of 2D shapes. Both implement the "Affine" class [26], which performs a linear mapping from 2D / 3D coordinates to other 2D / 3D coordinates.

The following JDORFX commands and their arguments for transformations have been implemented:

"*rotate3DShape*":

- "Double *angle*": the angle of rotation.
- "Double *pivotX pivotY pivotZ*": the 3D pivot point around which the shape is being rotated. Coordinates of 0 0 0 set the pivot point to the center of the shape.

- “Double *axisX axisY axisZ*”: defines the coordinate magnitude of the rotation axis. Values of 0 0 1 rotates the shape on the z axis, while values of 1 1 0 rotates the shape on the x axis and y axis by the same extent.

“*scale3DShape*” or “*scaleShape3D*”:

- “Double *sx sy sz*”: the scale factor of the coordinates. Values of 0 1.5 0 scales the shape on the y axis by 1.5.
- “Double *pivotX pivotY pivotZ*”: the 3D pivot point about which the scale occurs. Coordinates of 0 0 0 set the pivot point to the center of the shape.

“*shear3DShape*” or “*shearShape3D*”:

- “Double *xy yx*”: the shear coordinates.
- “Double *pivotX pivotY*”: the pivot point about which the shear occurs.

“*translate3DShape*” or “*translateShape3D*”:

- “Double *tx ty tz*”: moves the shape according to the values on each axis.

Fig. 19 shows all possible transformations set to 3D shapes: translate, shear, rotate and scale.

```
-- create JDORFX handler
-- load and add the Java Rexx command handler, using default name: JDORFX
call addJdorFXHandler
-- set default environment to JDORFX
address jdorfx

say "Create a scene and add and transform 3D shapes"
-- creating a new window with size 500 x 500
newimage 500 500
-- create new boxes
shape3d box1 box 360 280 0 200 20 200
shape3d box2 box 140 280 0 200 20 200
shape3d floor box 250 0 (-100) 500 20 500
--move box "floor" by x, y, z
translate3dshape floor 0 420 0
-- shear boxes with shx shy pivotX pivotY
shear3dshape box1 1 0 0 0
shear3dshape box2 (-1) 0 0 0
-- rotate shape  90 degrees on the z axis with its center as pivot point
rotate3dshape box1 90 0 0 0 0 0 1
-- rotate shape -90 degrees on the z axis with its center as pivot point
rotate3dshape box2 (-90) 0 0 0 0 0 1
-- create new cylinder
```

```
shape3d mySphere sphere 245 255 0 25
--scale the y axis of shape, pivot point is center of shape
scale3dshape mySphere 1 1.5 1 0 0 0
-- create new cylinder
shape3d cylinder1 cylinder 225 350 0 2 200
--rotate shape 45 degrees on the y axis, pivot point is 20px left of shape center
rotate3dshape cylinder1 45 20 0 0 0 1 0
--rotate shape 30 degrees on the z axis with its center as pivot point
rotate3dshape cylinder1 30 0 0 0 0 0 1
-- create new cylinder and apply rotation
shape3d cylinder2 cylinder 225 350 0 2 200
rotate3dshape cylinder2 90 20 0 0 0 1 0
rotate3dshape cylinder2 30 0 0 0 0 0 1
-- create new cylinder and apply rotation
shape3d cylinder3 cylinder 225 350 0 2 200
rotate3dshape cylinder3 135 20 0 0 0 1 0
rotate3dshape cylinder3 30 0 0 0 0 0 1
-- create new cylinder and apply rotation
shape3d cylinder4 cylinder 225 350 0 2 200
rotate3dshape cylinder4 180 20 0 0 0 1 0
rotate3dshape cylinder4 30 0 0 0 0 0 1
-- create new cylinder and apply rotation
shape3d cylinder5 cylinder 225 350 0 2 200
rotate3dshape cylinder5 225 20 0 0 0 1 0
rotate3dshape cylinder5 30 0 0 0 0 0 1
-- create new cylinder and apply rotation
shape3d cylinder6 cylinder 225 350 0 2 200
rotate3dshape cylinder6 270 20 0 0 0 1 0
rotate3dshape cylinder6 30 0 0 0 0 0 1
-- create new cylinder and apply rotation
shape3d cylinder7 cylinder 225 350 0 2 200
rotate3dshape cylinder7 315 20 0 0 0 1 0
rotate3dshape cylinder7 30 0 0 0 0 0 1
-- create new cylinder and apply rotation
shape3d cylinder8 cylinder 225 350 0 2 200
rotate3dshape cylinder8 360 20 0 0 0 1 0
rotate3dshape cylinder8 30 0 0 0 0 0 1
-- add shapes to scene
-- the order of placing shapes onto scene matters
-- later added shapes will cover previous ones, regardless of its distance to the camera
fillshape3d floor
fillshape3d box1
fillshape3d box2
fillshape3d cylinder4
```

```
fillshape3d cylinder5
fillshape3d cylinder6
fillshape3d cylinder7
fillshape3d cylinder8
fillshape3d mySphere
fillshape3d cylinder1
fillshape3d cylinder2
fillshape3d cylinder3
-- create new perspective camera and point light and add to scene
camera perspectiveCamera perspective 0 0 (-100)
setcamera  perspectiveCamera
light myLight point 100 50 (-400) white
setlight myLight
-- show created window
winshow
say
say "Sleep for 5 seconds and end program..."
sleep 5

-- get ooRexx-Java bridge, contains JDORFX Rexx command handler
::requires "jdorfx.CLS"
```

Listing 14. jdorfx_transform3d.rxj

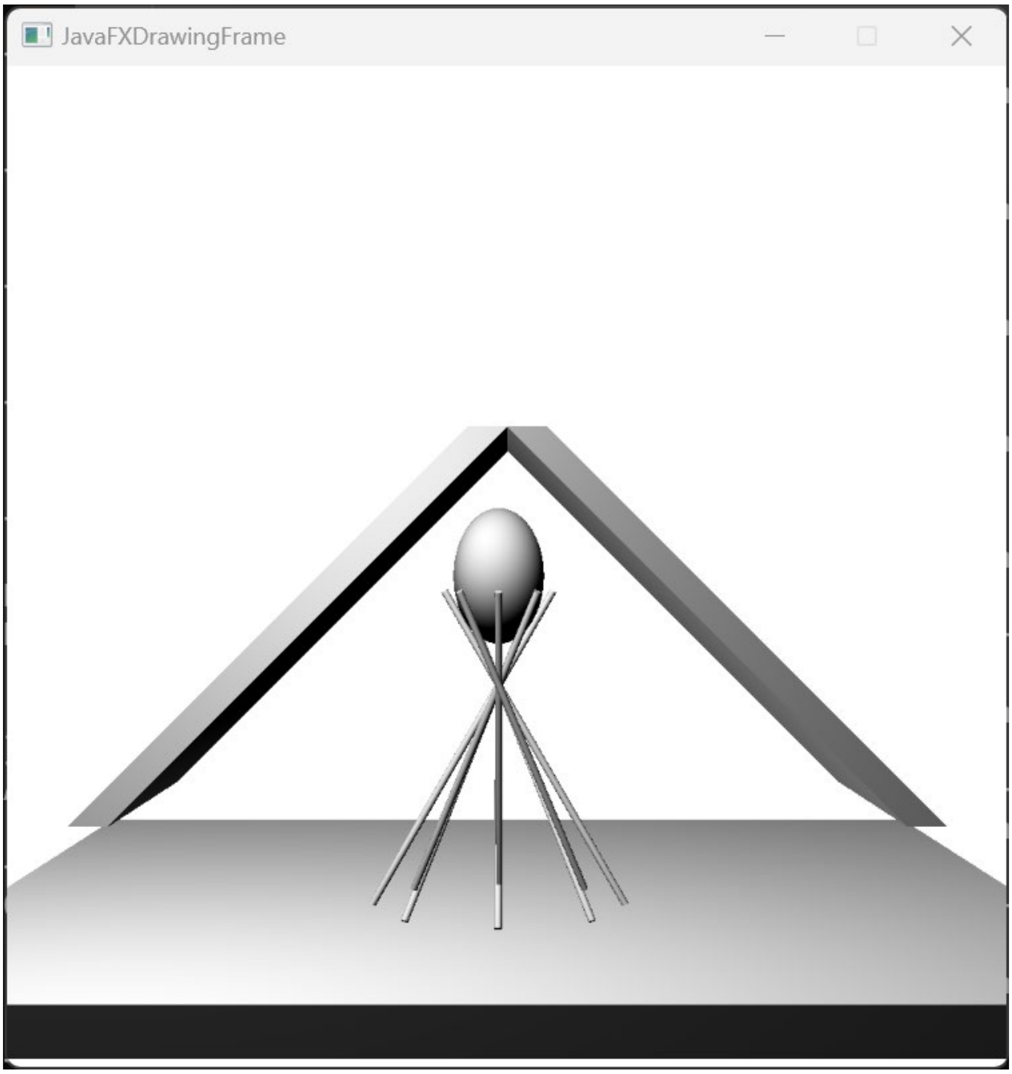

Fig. 22. Output jdorfx_transform3d.rxj

5.9. Material

The last example shows the implementation of the “PhongMaterial” [41] class. A “PhongMaterial” can be added to shapes as surfaces, with its properties determining the interaction with a light source.

The following properties of a “PhongMaterial” object can be set:

- “BumpMap”: acts as a normal map as RGB image. Combined with a diffuseMap, adds depth to the surface image and increases the appearance of a 3-dimensional surface.
- “DiffuseColor”: the color property of the surface.
- “DiffuseMap”: uses an image as the surface of a material.

- "SelfIlluminationMap": an image which is used to create a self illumination effect of a material. Brighter pixels of the image let these spots shine brighter on the material surface.
- "SpecularColor": specular color property of a material. It represents the color value of light that is reflecting from a surface.
- "SpecularMap": an image which defines the reflection properties of a material. Brighter pixels of the image let these spots reflect light better on the surface.
- "SpecularPower": defines the concentration or spread of highlighted reflection of a material. The smaller the value, the narrower the reflections and the smoother the surface appears.

The command "*material*" creates a new material and accepts the following arguments:

- "String *nickName*": set a name for the newly created material.
- "String *color*" [optional]: sets the "diffuseColor" of the material. If this argument is omitted, the color will be set to white as default.

"*materialColor*" sets the color properties of a material:

- "String *nickName*": targets the material with provided name.
- "String *type*": chooses which color property should be changed (*diffuseColor / diffuse* or *specularColor / specular*).
- "String *color*": sets the color of the color type.
- "Double *specularPower*" [only for type specular] [optional]: sets the power value of the "specularColor". If this argument is omitted, the value will be set to its default of 32.

"*materialMap*" adds an image as map to a material:

- "String *nickName*": targets the material with provided name.
- "String *type*": chooses which map type the image will be set as (*bump / bumpMap, diffuse / diffuseMap*, *selfillumination / selfilluminationMap or specular / specularMap*).
- "String *image*": select image via its file path, which will be added to the material as its map type. The supported image formats [50] are "BMP", "GIF", "JPEG", "PNG".

"*setMaterial*" adds a material to a shape:

- "String *shapeName*": targets shape with provided name.
- "String *materialName*": sets the named material to the shape.

The following example shows the use of different material properties. The image textures of the materials have been downloaded from the following website and are licensed as “CC0 1.0 DEED” [51]:

https://3dtextures.me/ [52]

```
-- create JDORFX handler
-- load and add the Java Rexx command handler, using default name: JDORFX
call addJdorFXHandler
-- set default environment to JDORFX
address jdorfx

say "Create a new scene with 3D shapes and materials"
-- creating a new window with size 500 x 500
newimage 500 500
-- create different shapes
shape3d rockBox box 100 100 0 100 100 100
shape3d furSphere sphere 250 100 0 50
shape3d colCylinder cylinder 400 100 0 20 100
shape3d tileSphere1 sphere 100 350 0 50
shape3d tileBox box 250 350 0 100 100 100
shape3d tileSphere2 sphere 400 350 0 50
-- add shapes to scene
fill3dshape rockBox
fill3dshape furSphere
fill3dshape colCylinder
fill3dshape tileSphere1
fill3dshape tileBox
fill3dshape tileSphere2
-- create new material named rock
material rock
-- add image with pathname "rock_diffuse.jpg" as diffuse map to material rock
materialmap rock diffuse "rock_diffuse.jpg"
-- add material rock to rockBox
setmaterial rockBox rock
--create new material fur
material fur
-- add image with pathname "fur_bump.jpg" as bump map to material fur
materialmap fur bump "fur_bump.jpg"
-- add image with pathname "fur_diffuse.jpg" as diffuse map to material rock
materialmap fur diffuse "fur_diffuse.jpg"
-- add material fur to furSphere
setmaterial furSphere fur
-- create new material colBlue and set color to blue
```

```
material colBlue blue
-- add material colBlue to colCylinder
setmaterial colCylinder colBlue
-- create new material tileDiff and set color to red
material tileDiff red
-- add image with pathname "tiles.jpg" as diffuse map to material tileDiff
materialmap tileDiff diffuse "tiles.jpg"
-- add material tileDiff to tileBox
setmaterial tileBox tileDiff
-- create new material tileSpec and set color to black
material tileSpec black
-- set material specularColor to white with specularPower of 10
materialcolor tileSpec specular white 10
-- add image with pathname "tiles.jpg" as specular map to material rock
materialmap tileSpec specular "tiles.jpg"
-- add material tileSpec to tileSphere1
setmaterial tileSphere1 tileSpec
-- create new material tileIll and set color to red
material tileIll red
-- add image with pathname "tiles.jpg" as selfillumination map to material tileIll
materialmap tileIll selfillumination "tiles.jpg"
-- add material tileIll to tileSphere2
setmaterial tileSphere2 tileIll
-- create perspective camera and add to scene
camera camera perspective 0 0 0
setcamera  camera
-- show created window
winshow
say
say "Sleep for 5 seconds and end program..."
sleep 5

-- get ooRexx-Java bridge, contains JDORFX Rexx command handler
::requires "jdorfx.CLS"
```

Listing 15. jdorfx_material.rxj

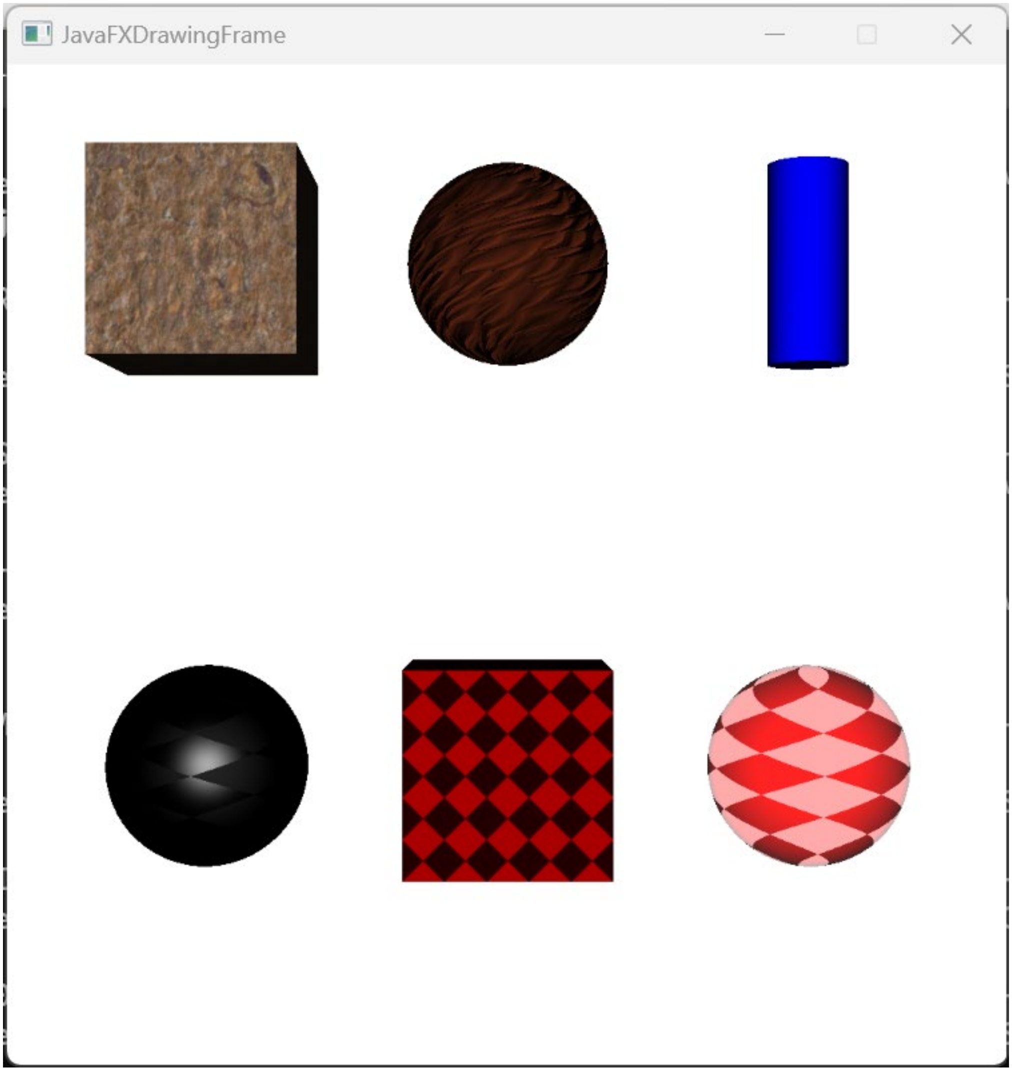

Fig. 23. Output jdorfx_material.rxj

5.10. Map

Texture images can be effectively utilized as material maps for 3D shapes to enhance the appearance of rendered objects. However, this approach presents limitations when pictures or logos are used, especially when they are applied to spheres or cylinders. In such cases, the images may appear distorted, due to the wrapping of the image around the shape's surface. Furthermore, if the image contains fully transparent pixels, the corresponding area of the shape will not be visible, and the shape of the rendered object will be unrecognisable. Fig. 21 illustrates this scenario by applying a PNG image of an ooRexx logo to all basic 3D shapes. While the pictures on the box' surface generally maintain their form, the ones on the sphere and cylinder are heavily distorted. Further, the transparent pixels of the image are transferred to the rendered objects, resulting in obscured edges and overall shape.

```
-- create JDORFX handler
-- load and add the Java Rexx command handler, using default name: JDORFX
call addJdorFXHandler
-- set default environment to JDORFX
address jdorfx

say "Create a new scene with 3D shapes and png maps"
-- creating a new window with size 500 x 500
newimage 500 500
-- create different 3D shapes
shape3d myBox box 150 150 0 100 100 100
shape3d myCylinder cylinder 250 350 0 40 100
shape3d mySphere sphere 350 150 0 50
-- add shapes to scene
fill3dshape myBox
fill3dshape mySphere
fill3dshape myCylinder
-- rotate myBox and myCylinder
rotate3dshape myBox (-20) 0 0 0 1 0 0
rotate3dshape myBox 60 0 0 0 0 1 0
rotate3dshape myCylinder 60 0 0 0 0 0 1
rotate3dshape myCylinder (-20) 0 0 0 1 0 0
-- create perspective camera and add to scene
camera camera perspective 0 0 0 50
setcamera  camera
-- create new material myMaterial and add png as diffuse map
material myMaterial
materialmap myMaterial diffuse "oorexx_256.png"
 -- add myMaterial to shapes
setmaterial myBox myMaterial
setmaterial mySphere myMaterial
setmaterial myCylinder myMaterial
-- show created window
winshow
say
say "Sleep for 5 seconds and end program..."
sleep 5

-- get ooRexx-Java bridge, contains JDORFX Rexx command handler
::requires "jdorfx.CLS"
```

Listing 16. jdorfx_map_unedited.rxj

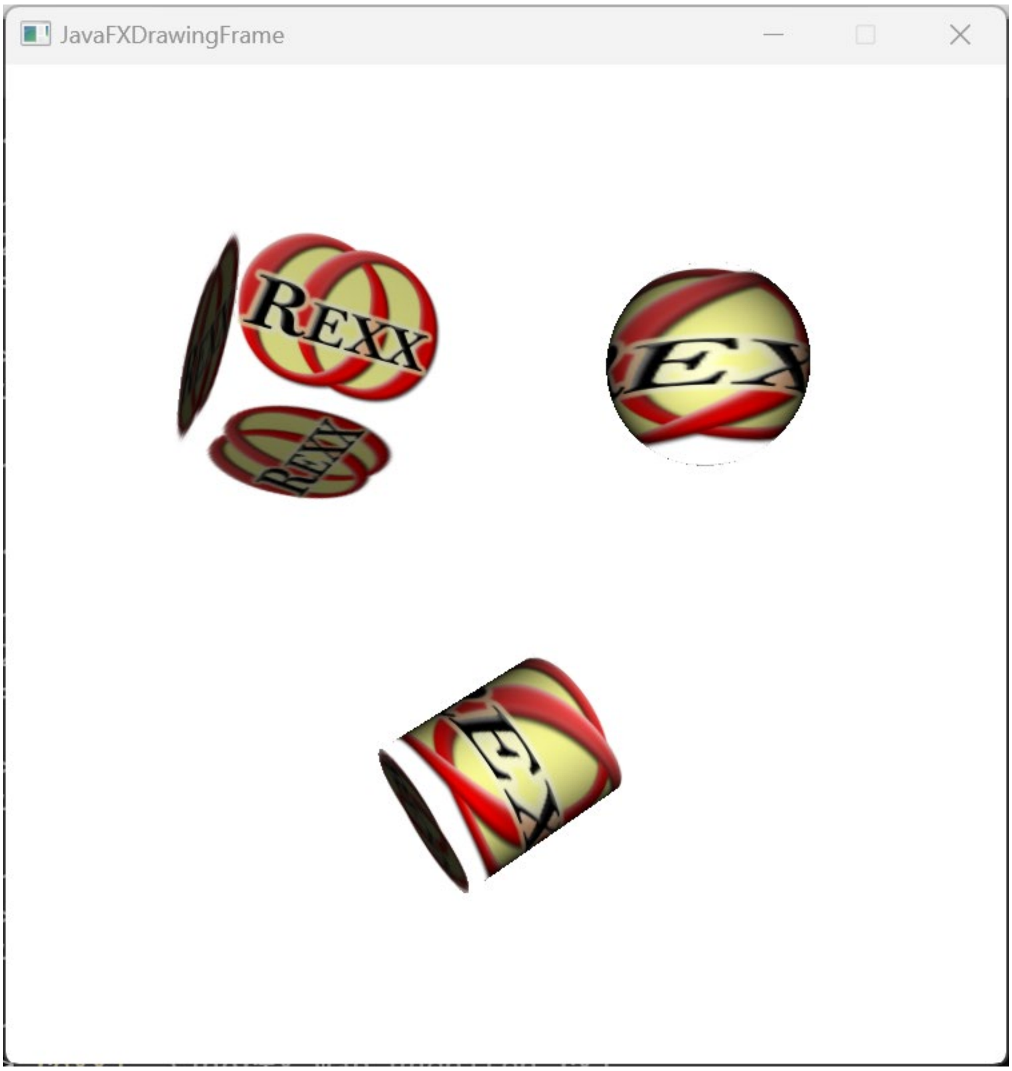

Fig. 24. Output jdorfx_map_unedited.rxj

In order to mitigate these issues, one solution is to edit the image which is used as a map within the program itself and store the new image in a variable. Behind the scene, JDORFX uses the Java class "PixelReader" [53] to read the color of each pixel of the supplied image and writes them via an instance of "PixelWriter" [54] onto a new "WritableImage" [55], which will be used as the new map. This functionality is achievable with the command "map", which creates a new material and accepts the following arguments:

- "String *nickName*": set a name for the image that will be saved.
- "String *imagePath*": the file path of the image.

The following additional arguments can be supplied to edit the image:

- "Integer *addWidth*": width that will be added to the size of the original image. If the value is set as a negative number, the image will be cropped. If no color argument is supplied, the added pixels will be transparent.

- “Integer *addHeight*”: height that will be added to the size of the original image. If the value is set as a negative number, the image will be cropped. If no color argument is supplied, the added pixels will be transparent.
- “Double *rotation*”: the angle of rotation. The image will be rotated clockwise with its centre as pivot point.
- “String *Color*” [optional]: If the supplied image contains transparent pixels or width and height have been added, the corresponding pixels on the material will be filled with the set color value.

It is worth noting that rotating an image beyond a magnitude of 90 degrees may result in a loss of quality. The rotated image may contain transparent pixels scattered across its surface, which will be filled with an optionally supplied color value. The following example shows how PNG images of ooRexx logos are edited and applied to box, sphere and cylinder objects. For spheres and cylinders, it is essential to set “addWidth” to be significantly larger than “addHeight” to preserve the original proportions of the images.

```
-- create JDORFX handler
-- load and add the Java Rexx command handler, using default name: JDORFX
call addJdorFXHandler
-- set default environment to JDORFX
address jdorfx

say "Create a new scene with 3D shapes and edited png maps"
-- creating a new window with size 500 x 500
newimage 500 500
-- create different 3D shapes
shape3d box1 box 100 100 0 100 100 100
shape3d box2 box 100 250 0 100 100 100
shape3d box3 box 100 400 0 100 100 100
shape3d sphere1 sphere 250 100 0 50
shape3d sphere2 sphere 250 250 0 50
shape3d sphere3 sphere 250 400 0 50
shape3d cylinder1 cylinder 400 100 0 40 100
shape3d cylinder2 cylinder 400 250 0 40 100
shape3d cylinder3 cylinder 400 400 0 40 100
-- add shapes to scene
fill3dshape box1
fill3dshape box2
fill3dshape box3
fill3dshape sphere1
fill3dshape sphere2
fill3dshape sphere3
```

```
fill3dshape cylinder1
fill3dshape cylinder2
fill3dshape cylinder3
-- rotate shapes
rotate3dshape box1 (-20) 0 0 0 1 1 0
rotate3dshape box1 60 0 0 0 0 1 0
rotate3dshape box2 60 0 0 0 0 1 0
rotate3dshape box3 20 0 0 0 1 1 0
rotate3dshape box3 (-60) 0 0 0 0 1 0
rotate3dshape cylinder1 40 0 0 0 0 0 1
rotate3dshape cylinder2 70 0 0 0 1 0 1
rotate3dshape cylinder2 30 0 0 0 0 1 0
rotate3dshape cylinder3 (-40) 0 0 0 0 0 1
-- create perspective camera and add to scene
camera camera perspective 0 0 0 50
setcamera  camera
-- create new materials for each shape
material boxMaterial1
material boxMaterial2
material boxMaterial3
material sphereMaterial1
material sphereMaterial2
material sphereMaterial3
material cylinderMaterial1
material cylinderMaterial2
material cylinderMaterial3
-- create new images with name "boxMap1" based on image with image path "oorexx_256.png"
-- add width=50 and height=50 to image, set rotation angle=0 and set a color="cyan"
map boxMap1 "oorexx_256.png" 50 50 0 cyan
map boxMap2 "oorexx4ooo_256.png" 50 50 90 blue
map boxMap3 "bsf4oorexx_256.png" 50 50 180 green
map sphereMap1 "oorexx_256.png" 800 200 20 white
map sphereMap2 "oorexx4ooo_256.png" 800 200 0 yellow
map sphereMap3 "bsf4oorexx_256.png" 800 200 340 orange
map cylinderMap1 "oorexx_256.png" 600 200 0 pink
map cylinderMap2 "oorexx4ooo_256.png" 500 200 270 magenta
map cylinderMap3 "bsf4oorexx_256.png" 600 200 0 red
-- set maps as diffuse maps to each material
materialmap boxMaterial1 diffuse boxMap1
materialmap boxMaterial2 diffuse boxMap2
materialmap boxMaterial3 diffuse boxMap3
materialmap sphereMaterial1 diffuse sphereMap1
materialmap sphereMaterial2 diffuse sphereMap2
materialmap sphereMaterial3 diffuse sphereMap3
```

```
materialmap cylinderMaterial1 diffuse cylinderMap1
materialmap cylinderMaterial2 diffuse cylinderMap2
materialmap cylinderMaterial3 diffuse cylinderMap3
--set materials to shapes
setmaterial box1 boxMaterial1
setmaterial box2 boxMaterial2
setmaterial box3 boxMaterial3
setmaterial sphere1 sphereMaterial1
setmaterial sphere2 sphereMaterial2
setmaterial sphere3 sphereMaterial3
setmaterial cylinder1 cylinderMaterial1
setmaterial cylinder2 cylinderMaterial2
setmaterial cylinder3 cylinderMaterial3
-- show created window
winshow
say
say "Sleep for 5 seconds and end program..."
sleep 5

-- get ooRexx-Java bridge, contains JDORFX Rexx command handler
::requires "jdorfx.CLS"
```

Listing 17. jdorfx_map_edited.rxj

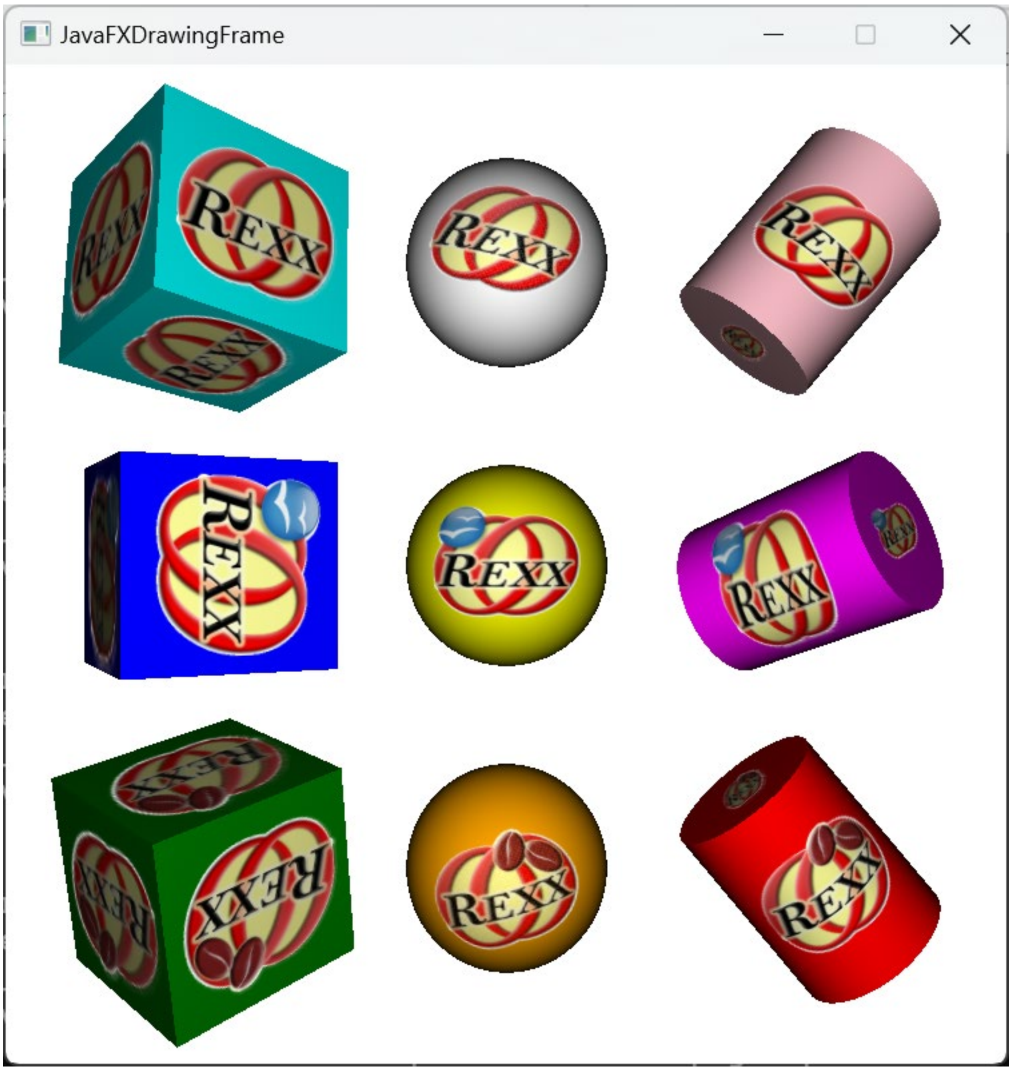

Fig. 25. Output jdorfx_map_edited.rxj

6. Limitations

While JDORFX demonstrates promising results in terms of functionality, it is important to acknowledge its limitations as a first version. Firstly, JDORFX does not encompass all commands available in JDOR, such as animations, the use of gradient paint or saving the GUI output as images. Furthermore, despite its capabilities, JDORFX's architecture can be improved upon to optimize performance, which is especially relevant within 3D graphics rendering. Running long programs may lead to blocking of the UI and making it unresponsive. Additionally, while JDORFX provides an extensive set of features for 2D and 3D graphics, there is still room for expansion and refinement. Future iterations of JDORFX could benefit from increasing its graphical functionalities.

7. Conclusion

In this thesis, the development of JDORFX is described, a JavaFX-based graphics framework, and compared to its counterpart JDOR, which utilizes awt based Java2D classes. The goal was to provide a JavaFX GUI for ooRexx programmers, which uses the same input commands as JDOR. While different in their architectures, it has been shown that functionalities and GUI output are almost identical for both frameworks aside from color tones.

A significant advantage of JDORFX over JDOR is its newly implemented support for 3D graphics rendering. Nutshell examples have shown the functionalities of ooRexx commands, enabling the creation and transformation of 3D shapes, camera objects and light objects. This is the first time ooRexx programmers are able to utilize JavaFX 3D graphics without the need to learn about their classes or Java code itself.

Looking ahead, future developments in JDORFX could focus on enhancing its 3D graphics capabilities, improving performance optimization and expanding on the wide variety of JavaFX competences.

References

[1] R. G. Flatscher, "202209_JDOR_command_handler," [Online]. Available: https://www.rexxla.org/presentations/2022/202209_JDOR_command_handler.pdf (accessed: Oct. 20 2023).

[2] Oracle, *1 JavaFX Overview (Release 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm (accessed: Oct. 20 2023).

[3] R. G. Flatscher, "202303_ISECON23_Flatscher_Proposing_ooRexx," [Online]. Available: https://www.rexxla.org/presentations/2023/202303_ISECON23_Flatscher_Proposing_ooRexx.pdf (accessed: Oct. 20 2023).

[4] Rexx Language Association, *Open Object Rexx.* [Online]. Available: https://www.oorexx.org/about.html (accessed: Oct. 16 2023).

[5] www.javatpoint.com, *History of Java - Javatpoint.* [Online]. Available: https://www.javatpoint.com/history-of-java (accessed: Oct. 16 2023).

[6] www.javatpoint.com, *Features of Java - Javatpoint.* [Online]. Available: https://www.javatpoint.com/features-of-java (accessed: Oct. 16 2023).

[7] C. Murcko, V. Orlikowski, and R. G. Flatscher, *Apache Commons BSF™ - Bean Scripting Framework.* [Online]. Available: https://commons.apache.org/proper/commons-bsf/ (accessed: Oct. 16 2023).

[8] Oracle, *JFrame (Java Platform SE 8).* [Online]. Available: https://docs.oracle.com/javase/8/docs/api/javax/swing/JFrame.html (accessed: Nov. 2 2023).

[9] Oracle, *JPanel (Java Platform SE 8).* [Online]. Available: https://docs.oracle.com/javase/8/docs/api/javax/swing/JPanel.html (accessed: Nov. 2 2023).

[10] Oracle, *Graphics2D (Java Platform SE 8).* [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics2D.html (accessed: Feb. 2 2024).

[11] Oracle, *BufferedImage (Java Platform SE 8).* [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/awt/image/BufferedImage.html (accessed: Nov. 2 2023).

[12] Oracle, *Shape (Java Platform SE 8).* [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/awt/Shape.html (accessed: Feb. 2 2024).

[13] Oracle, *Oracle Java SE Support Roadmap.* [Online]. Available: https://www.oracle.com/java/technologies/java-se-support-roadmap.html (accessed: Apr. 15 2024).

[14] BellSoft Corporation, *Java Download | Java 8, Java 11, Java 17, Java 21 - OpenJDK Builds for Linux, Windows & macOS.* [Online]. Available: https://bell-sw.com/pages/downloads/#jdk-8-lts (accessed: Mar. 2 2024).

[15] Sourceforge, *Download ooRexx-5.0.0-12583.windows.x86_64.exe (ooRexx (Open Object Rexx)).* [Online]. Available: https://sourceforge.net/projects/oorexx/files/oorexx/5.0.0/ooRexx-5.0.0-12583.windows.x86_64.exe/download (accessed: Mar. 2 2024).

[16] Sourceforge, *Download BSF4ooRexx.* [Online]. Available: https://sourceforge.net/projects/bsf4oorexx/files/latest/download (accessed: Mar. 2 2024).

[17] Oracle, *Application (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/application/Application.html (accessed: Oct. 25 2024).

[18] Oracle, *Stage (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/stage/Stage.html (accessed: Oct. 25 2023).

[19] Oracle, *Scene (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/Scene.html (accessed: Oct. 25 2023).

[20] Oracle, *Pane (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/Pane.html (accessed: Jan. 20 2024).

[21] Oracle, *Group (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/Group.html (accessed: Jan. 20 2024).

[22] Oracle, *ConcurrentLinkedDeque (Java Platform SE 8).* [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentLinkedDeque.html (accessed: Dec. 20 2023).

[23] Oracle, *Runnable (Java Platform SE 7).* [Online]. Available: https://docs.oracle.com/javase%2F7%2Fdocs%2Fapi%2F%2F/java/lang/Runnable.html (accessed: Dec. 10 2024).

[24] Oracle, *The switch Statement (The Java™ Tutorials > Learning the Java Language > Language Basics).* [Online]. Available: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html (accessed: Jan. 2 2024).

[25] Oracle, *AffineTransform (Java Platform SE 7).* [Online]. Available: https://docs.oracle.com/javase%2F7%2Fdocs%2Fapi%2F%2F/java/awt/geom/AffineTransform.html (accessed: Jan. 14 2024).

[26] Oracle, *Affine (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/transform/Affine.html (accessed: Feb. 2 2024).

[27] Oracle, *GraphicsContext (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/canvas/GraphicsContext.html (accessed: Jan. 2 2024).

[28] Oracle, *Canvas (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/canvas/Canvas.html (accessed: Mar. 2 2024).

[29] Oracle, *Shape (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Shape.html (accessed: Jan. 2 2024).

[30] W3 Schools, *Java Packages.* [Online]. Available: https://www.w3schools.com/java/java_packages.asp (accessed: Feb. 24 2024).

[31] Oracle, *Using JAR Files: The Basics (The Java™ Tutorials > Deployment > Packaging Programs in JAR Files).* [Online]. Available: https://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html (accessed: May 6 2024).

[32] R. G. Flatscher, "202403-01_JDOR," [Online]. Available: https://www.rexxla.org/presentations/2024/202403-01_JDOR.pdf (accessed: May 6 2024).

[33] R. G. Flatscher, "202403-04_Releasing_BSF4ooRexx850," [Online]. Available: https://www.rexxla.org/presentations/2024/202403-04_Releasing_BSF4ooRexx850.pdf (accessed: May 6 2024).

[34] www.javatpoint.com, *Java Color Codes - Javatpoint.* [Online]. Available: https://www.javatpoint.com/java-color-codes (accessed: Feb. 2 2024).

[35] Oracle, *Color (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/paint/Color.html#GREEN (accessed: Feb. 2 2024).

[36] Oracle, *HashMap (Java Platform SE 8).* [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html (accessed: Feb. 5 2024).

[37] Oracle, *Box (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Box.html (accessed: Feb. 5 2024).

[38] Oracle, *Cylinder (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Cylinder.html (accessed: Feb. 5 2024).

[39] Oracle, *Sphere (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Sphere.html (accessed: Feb. 5 2024).

[40] Oracle, *Shape3D (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Shape3D.html (accessed: Feb. 5 2024).

[41] Oracle, *PhongMaterial (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/paint/PhongMaterial.html (accessed: Feb. 20 2024).

[42] Oracle, *DrawMode (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/DrawMode.html (accessed: Feb. 5 2024).

[43] Oracle, *CullFace (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/CullFace.html (accessed: Feb. 5 2024).

[44] Oracle, *Camera (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/Camera.html (accessed: Feb. 7 2024).

[45] Oracle, *ParallelCamera (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/ParallelCamera.html (accessed: Feb. 7 2024).

[46] Oracle, *PerspectiveCamera (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/PerspectiveCamera.html (accessed: Feb. 7 2024).

[47] Oracle, *LightBase (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/LightBase.html (accessed: Feb. 7 2024).

[48] Oracle, *AmbientLight (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/AmbientLight.html (accessed: Feb. 7 2024).

[49] Oracle, *PointLight (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/PointLight.html (accessed: Feb. 7 2024).

[50] Oracle, *Image (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/Image.html (accessed: Feb. 20 2024).

[51] Creative Commons, *CC0 1.0 Deed | CC0 1.0 Universal | Creative Commons.* [Online]. Available: https://creativecommons.org/publicdomain/zero/1.0/ (accessed: Feb. 5 2024).

[52] 3D TEXTURES, *3D TEXTURES.* [Online]. Available: https://3dtextures.me/ (accessed: Feb. 2 2024).

[53] Oracle, *PixelReader (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/PixelReader.html (accessed: Apr. 29 2024).

[54] Oracle, *PixelWriter (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/PixelWriter.html (accessed: Apr. 29 2024).

[55] Oracle, *WritableImage (JavaFX 8).* [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/WritableImage.html (accessed: Apr. 29 2024).