```
#feature-id   StarStretch : SetiAstro > Star Stretch
#feature-info This script performs a stretch of a linear star image, boosts color, and
removes any green cast.

/******************************************************************************

*#####################################################################
#
 *#        ___     __      ___       __                              #
 *#       / __/___/ /__   / _ | ___ / /________                        #
 *#      _\ \/ -_) _ _   / __ |(_-</ __/ __/ _ \                    #
 *#     /___/\__/_//_/  /_/ |_/___/\__/_/  \___/                      #
 *#                                                       #

*#####################################################################
#
 * Star Stretch Script
 * Version: 2.6
 * Author: Franklin Marek
 * Website: www.setiastro.com
 *
 * This script performs a stretch of a linear star image, boosts color, and removes any
green cast.
 *
 * This work is licensed under a Creative Commons Attribution-NonCommercial 4.0
International License.
 * To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/4.0/
 *
 * You are free to:
 * 1. Share — copy and redistribute the material in any medium or format
 * 2. Adapt — remix, transform, and build upon the material
 *
 * Under the following terms:
 * 1. Attribution — You must give appropriate credit, provide a link to the license, and
indicate if changes were made. You may do so in any reasonable manner, but not in
any way that suggests the licensor endorses you or your use.
 * 2. NonCommercial — You may not use the material for commercial purposes.
 *
 * @license CC BY-NC 4.0 (http://creativecommons.org/licenses/by-nc/4.0/)
 *
 * COPYRIGHT © 2024 Franklin Marek. ALL RIGHTS RESERVED.
 ******************************************************************************/

#include <pjsr/StdButton.jsh>
#include <pjsr/StdIcon.jsh>
#include <pjsr/StdCursor.jsh>
```

```
#include <pjsr/Sizer.jsh>
#include <pjsr/FrameStyle.jsh>
#include <pjsr/NumericControl.jsh>

// include constants
#include <pjsr/ImageOp.jsh>
#include <pjsr/SampleType.jsh>
#include <pjsr/UndoFlag.jsh>

#define VERSION "v2.6"

// define a global variable containing the script's parameters
var StarStretchParameters = {
  amount: 5,
  targetView: undefined,
  satAmount: 1, // Default saturation amount
  removeGreen: false, // Default SCNR option
  showPreview: false, // Default Show Preview option
  saturationLevel: [
    [0.00000, 0.40000],
    [0.50000, 0.70000],
    [1.00000, 0.40000]
  ],

  // stores the current parameters values into the script instance
  save: function() {
    Parameters.set("amount", StarStretchParameters.amount);
    Parameters.set("satAmount", StarStretchParameters.satAmount);
    Parameters.set("removeGreen", StarStretchParameters.removeGreen);
    Parameters.set("showPreview", StarStretchParameters.showPreview);
  },

  // loads the script instance parameters
  load: function () {
    if (Parameters.has("amount"))
      StarStretchParameters.amount = Parameters.getReal("amount");
    if (Parameters.has("satAmount"))
      StarStretchParameters.satAmount = Parameters.getReal("satAmount");
    if (Parameters.has("removeGreen"))
      StarStretchParameters.removeGreen = Parameters.getBoolean("removeGreen");
    if (Parameters.has("showPreview"))
      StarStretchParameters.showPreview = Parameters.getBoolean("showPreview");
  },

  // Update saturation level matrix based on satAmount
  updateSaturationLevel: function() {
      var satAmount = StarStretchParameters.satAmount;
      StarStretchParameters.saturationLevel = [
        [0.00000, satAmount * 0.40000],
        [0.50000, satAmount * 0.70000],
        [1.00000, satAmount * 0.40000]
      ];
   }
}

function applyPixelMath(view, amount) {
   // Instantiate the PixelMath process
   var P = new PixelMath;
   P.expression = "((3^" + amount + ")*$T)/((3^" + amount + " - 1)*$T + 1)";

   // Perform the pixel math transformation
   P.executeOn(view);
}

function applyColorSaturation(view, satAmount) {
   // Instantiate the ColorSaturation process
   var P = new ColorSaturation;
   P.HS = StarStretchParameters.saturationLevel;
   P.HSt = ColorSaturation.prototype.AkimaSubsplines;
   P.hueShift = 0.000;

   // Perform the colorsaturation transformation
   P.executeOn(view);
}


function applySCNR(view) {
   var P = new SCNR;
   P.amount = 1.00;
   P.protectionMethod = SCNR.prototype.AverageNeutral;
   P.colorToRemove = SCNR.prototype.Green;
   P.preserveLightness = true;

   P.executeOn(view);
}

// ScrollControl class definition
function ScrollControl(parent) {
    this.__base__ = ScrollBox;
    this.__base__(parent);

    this.autoScroll = true;
    this.tracking = true;

    this.displayImage = null;
    this.dragging = false;
    this.dragOrigin = new Point(0);

    this.viewport.cursor = new Cursor(StdCursor_OpenHand);

    this.getImage = function() {
        return this.displayImage;
    };

    this.doUpdateImage = function(image) {
        this.displayImage = image;
        this.initScrollBars();
        this.viewport.update();
    };

    this.initScrollBars = function() {
        var image = this.getImage();
        if (image == null || image.width <= 0 || image.height <= 0) {
            this.setHorizontalScrollRange(0, 0);
            this.setVerticalScrollRange(0, 0);
        } else {
            this.setHorizontalScrollRange(0, Math.max(0, image.width - 
this.viewport.width));
            this.setVerticalScrollRange(0, Math.max(0, image.height - this.viewport.height));
        }
        this.viewport.update();
    };

    this.viewport.onResize = function() {
        this.parent.initScrollBars();
    };

    this.onHorizontalScrollPosUpdated = function(x) {
        this.viewport.update();
    };

    this.onVerticalScrollPosUpdated = function(y) {
        this.viewport.update();
    };

    this.viewport.onMousePress = function(x, y, button, buttons, modifiers) {
        this.cursor = new Cursor(StdCursor_ClosedHand);
        with (this.parent) {
            dragOrigin.x = x;
            dragOrigin.y = y;
            dragging = true;
        }
    };

    this.viewport.onMouseMove = function(x, y, buttons, modifiers) {
        with (this.parent) {
            if (dragging) {
                scrollPosition = new Point(scrollPosition).translatedBy(dragOrigin.x - x,
dragOrigin.y - y);
                dragOrigin.x = x;
                dragOrigin.y = y;
            }
        }
    };

    this.viewport.onMouseRelease = function(x, y, button, buttons, modifiers) {
        this.cursor = new Cursor(StdCursor_OpenHand);
        this.parent.dragging = false;
    };

    this.viewport.onPaint = function(x0, y0, x1, y1) {
        var g = new Graphics(this);
        var result = this.parent.getImage();
        if (result == null) {
            g.fillRect(x0, y0, x1, y1, new Brush(0xff000000));
        } else {
            result.selectedRect = (new Rect(x0, y0, x1,
y1)).translated(this.parent.scrollPosition);
            g.drawBitmap(x0, y0, result.render());
            result.resetRectSelection();
        }
        g.end();
        gc();
    };

    this.initScrollBars();
}
ScrollControl.prototype = new ScrollBox;

function StarStretchDialog() {
   this.__base__ = Dialog;
   this.__base__();

   this.title = new Label(this);
```

```
  this.title.text = "Star Stretch " + VERSION + ": Linear to Non-Linear Stretch\n of a
Stars Only Image";
  this.title.textAlignment = TextAlign_Center;
  this.title.styleSheet = "font-weight: bold; font-size: 14pt; background-color: #f0f0f0;";
  this.title.minHeight = 40;
  this.title.maxHeight = 50;

  this.description = new TextBox(this);
  this.description.text = "Please select your combined stars only image to stretch\n" +
    "Default value is a stretch of " + StarStretchParameters.amount + "\n" +
    "Default color boost value is " + StarStretchParameters.satAmount + "\n\n" +
    "Written by Franklin Marek. Copyright 2024";
  this.description.readOnly = true;
  this.description.backgroundColor = 0xd3d3d3; // Grey background
  this.description.maxHeight = 100;
  this.description.minWidth = 400;

  // Find the active window
  let activeWindow = ImageWindow.activeWindow;
  if (!activeWindow.isNull) {
    StarStretchParameters.targetView = activeWindow.mainView;
  } else {
    StarStretchParameters.targetView = null;
  }

  // add a view picker
  this.viewList = new ViewList(this);
  this.viewList.getAll();
  if (StarStretchParameters.targetView) {
    this.viewList.currentView = StarStretchParameters.targetView;
  }
  this.viewList.onViewSelected = function(view) {
    StarStretchParameters.targetView = view;
  };

  // create the input slider for Stretch Amount
  this.AmountControl = new NumericControl(this);
  this.AmountControl.label.text = "Stretch Amount:";
  this.AmountControl.label.width = 120;
  this.AmountControl.setRange(0, 8);
  this.AmountControl.setPrecision(2);
  this.AmountControl.slider.setRange(0, 100);
  this.AmountControl.setValue(StarStretchParameters.amount); // Set the default value
  this.AmountControl.toolTip = "<p>Adjust above 5 with caution.</p>";
  this.AmountControl.onValueUpdated = function(value) {
    StarStretchParameters.amount = value;
  };

  // create the input slider for Saturation Amount
  this.SaturationAmountControl = new NumericControl(this);
  this.SaturationAmountControl.label.text = "Color Boost Amount:";
  this.SaturationAmountControl.label.width = 120;
  this.SaturationAmountControl.setRange(0, 2);
  this.SaturationAmountControl.setPrecision(2);
  this.SaturationAmountControl.slider.setRange(0, 200);
  this.SaturationAmountControl.setValue(StarStretchParameters.satAmount); // Set the
default value
  this.SaturationAmountControl.toolTip = "<p>Adjust the color saturation amount.</
p>";
  this.SaturationAmountControl.onValueUpdated = function(value) {
    StarStretchParameters.satAmount = value;
    StarStretchParameters.updateSaturationLevel(); // Update the saturation level
matrix
  };

  // create the checkbox for SCNR
  this.SCNRCheckBox = new CheckBox(this);
  this.SCNRCheckBox.text = "Remove Green via SCNR (Optional)";
  this.SCNRCheckBox.checked = StarStretchParameters.removeGreen;
  this.SCNRCheckBox.toolTip = "<p>Enable or disable green cast removal using
SCNR.</p>";
  this.SCNRCheckBox.onCheck = function(checked) {
    StarStretchParameters.removeGreen = checked;
  };

  // create the checkbox for Show Preview
  this.showPreviewCheckBox = new CheckBox(this);
  this.showPreviewCheckBox.text = "Show Preview";
  this.showPreviewCheckBox.checked = StarStretchParameters.showPreview;
  this.showPreviewCheckBox.toolTip = "<p>Enable or disable preview of the stretch.</
p>";
  this.showPreviewCheckBox.onCheck = function(checked) {
    StarStretchParameters.showPreview = checked;
    this.previewControl.visible = checked;
    this.zoomSizer.visible = checked;
    if (checked && StarStretchParameters.targetView) {
      let selectedImage = StarStretchParameters.targetView.image;
      if (selectedImage) {
        let tmpImage = this.createAndDisplayTemporaryImage(selectedImage);
        this.previewControl.displayImage = tmpImage;
        this.previewControl.initScrollBars();
        this.previewControl.viewport.update();
```

```
            }
         }
         this.adjustToContents();
      }.bind(this);

      // Add create instance button
      this.newInstanceButton = new ToolButton(this);
      this.newInstanceButton.icon = this.scaledResource(":/process-interface/new-
   instance.png");
      this.newInstanceButton.setScaledFixedSize(24, 24);
      this.newInstanceButton.toolTip = "New Instance";
      this.newInstanceButton.onMousePress = () => {
         // stores the parameters
         StarStretchParameters.save();
         // create the script instance
         this.newInstance();
      };

      // Create zoom dropdown
      this.zoomLabel = new Label(this);
      this.zoomLabel.text = "Zoom: ";
      this.zoomLabel.textAlignment = TextAlign_Right | TextAlign_VertCenter;

      this.zoomComboBox = new ComboBox(this);
      this.zoomComboBox.addItem("1:1");
      this.zoomComboBox.addItem("1:2");
      this.zoomComboBox.addItem("1:4");
      this.zoomComboBox.addItem("1:8");
      this.zoomComboBox.addItem("Fit to Preview");
      this.zoomComboBox.currentItem = 2; // Set default to "1:4"
      this.zoomComboBox.minWidth = 120; // Set minimum width
      this.zoomComboBox.onItemSelected = (index) => {
         this.previewControl.zoomFactor = -Math.pow(2, index);
         if (StarStretchParameters.showPreview) {
            this.refreshPreview();
         }
      };

      // Create preview refresh button
      this.previewRefreshButton = new PushButton(this);
      this.previewRefreshButton.text = "Refresh Preview";
      this.previewRefreshButton.minWidth = 120; // Set minimum width
      this.previewRefreshButton.onClick = () => {
         this.refreshPreview();
      };
      // Create a sizer for zoom controls and refresh button
      this.zoomSizer = new HorizontalSizer;
      this.zoomSizer.margin = 4;
      this.zoomSizer.spacing = 4;
      this.zoomSizer.add(this.zoomLabel);
      this.zoomSizer.add(this.zoomComboBox);
      this.zoomSizer.addSpacing(12);
      this.zoomSizer.add(this.previewRefreshButton);

      // prepare the execution button
      this.execButton = new PushButton(this);
      this.execButton.text = "Execute";
      this.execButton.width = 80; // Increased width
      this.execButton.onClick = () => {
         this.applyStretchToMainImage();
         this.ok();
      };

      // create a horizontal slider to layout the execution button
      this.execButtonSizer = new HorizontalSizer;
      this.execButtonSizer.margin = 8;
      this.execButtonSizer.add(this.newInstanceButton)
      this.execButtonSizer.addSpacing(12);
      this.execButtonSizer.add(this.zoomSizer);
      this.execButtonSizer.addStretch();
      this.execButtonSizer.add(this.execButton)

      // Create a preview control
      this.previewControl = new ScrollControl(this);
      this.previewControl.setMinWidth(300);  // Set minimum width to avoid taking too
   much space
      this.previewControl.setMinHeight(300); // Set minimum height
      this.previewControl.visible = false;   // Hide preview control initially

      // layout the dialog
      this.leftSizer = new VerticalSizer;
      this.leftSizer.margin = 8;
      this.leftSizer.spacing = 8;
      this.leftSizer.add(this.title);
      this.leftSizer.addSpacing(8);
      this.leftSizer.add(this.description);
      this.leftSizer.addSpacing(8);
      this.leftSizer.add(this.viewList);
      this.leftSizer.addSpacing(8);
      this.leftSizer.add(this.AmountControl);
      this.leftSizer.addSpacing(8);
```

```
  this.leftSizer.add(this.SaturationAmountControl);
  this.leftSizer.addSpacing(8);
  this.leftSizer.add(this.SCNRCheckBox);
  this.leftSizer.addSpacing(8);
  this.leftSizer.add(this.showPreviewCheckBox);
  this.leftSizer.addSpacing(8);
  this.leftSizer.add(this.execButtonSizer);
  this.leftSizer.addStretch();
  this.leftSizer.minWidth = 300; // Set minimum width for left sizer

  this.mainSizer = new HorizontalSizer;
  this.mainSizer.margin = 8;
  this.mainSizer.spacing = 8;
  this.mainSizer.add(this.leftSizer);
  this.mainSizer.add(this.previewControl, 1, Align_Expand);  // Ensure preview control
gets allocated proper space

  this.sizer = this.mainSizer;

  this.adjustToContents();

  this.onShow = () => {
    this.previewControl.visible = false;
    this.zoomSizer.visible = false;
    this.adjustToContents();
  };

  this.createAndDisplayTemporaryImage = function(selectedImage) {
    let window = new ImageWindow(
      selectedImage.width, selectedImage.height,
      selectedImage.numberOfChannels,
      selectedImage.bitsPerSample,
      selectedImage.isReal,
      selectedImage.isColor
    );

    window.mainView.beginProcess();
    window.mainView.image.assign(selectedImage);
    window.mainView.endProcess();

    applyPixelMath(window.mainView, StarStretchParameters.amount);
    if (!isGrayscale(window.mainView)) {
      applyColorSaturation(window.mainView, StarStretchParameters.satAmount);
      if (StarStretchParameters.removeGreen) {
        applySCNR(window.mainView);
      }
    }

    var P = new IntegerResample;
    switch (this.zoomComboBox.currentItem) {
      case 0:
        P.zoomFactor = -1;
        break;
      case 1:
        P.zoomFactor = -2;
        break;
      case 2:
        P.zoomFactor = -4;
        break;
      case 3:
        P.zoomFactor = -8;
        break;
      case 4:
        const previewWidth = this.previewControl.width;
        const widthScale = Math.floor(selectedImage.width / previewWidth);
        P.zoomFactor = -Math.max(widthScale, 1);
        break;
      default:
        P.zoomFactor = -2;
        break;
    }

    P.executeOn(window.mainView);

    let resizedImage = new Image(window.mainView.image);

    if (resizedImage.width > 0 && resizedImage.height > 0) {
      this.previewControl.displayImage = resizedImage;
      this.previewControl.doUpdateImage(resizedImage);
      this.previewControl.initScrollBars();
    } else {
      console.error("Resized image has invalid dimensions.");
    }

    window.forceClose();

    return resizedImage;
  };

  this.adjustToContents();
}
```

```
StarStretchDialog.prototype = new Dialog;

function disableSTF(targetView) {
   // Create a new instance of the STF process
   var stf = new ScreenTransferFunction;

   // Set the STF parameters to disable (identity transformation)
   var stfParams = [ // c0, c1, m, r0, r1
      [0.00000, 1.00000, 0.50000, 0.00000, 1.00000],
      [0.00000, 1.00000, 0.50000, 0.00000, 1.00000],
      [0.00000, 1.00000, 0.50000, 0.00000, 1.00000],
      [0.00000, 1.00000, 0.50000, 0.00000, 1.00000]
   ];

   stf.STF = stfParams;
   stf.executeOn(targetView, false); // false to apply to the target view without a new
instance

   console.writeln("STF has been disabled.");
}

StarStretchDialog.prototype.applyStretchToMainImage = function() {
  if (StarStretchParameters.targetView) {
    applyPixelMath(StarStretchParameters.targetView, StarStretchParameters.amount);
    if (!isGrayscale(StarStretchParameters.targetView)) {
      applyColorSaturation(StarStretchParameters.targetView,
StarStretchParameters.satAmount);
      if (StarStretchParameters.removeGreen) {
        applySCNR(StarStretchParameters.targetView);
      }
    }
    console.show();
    disableSTF(StarStretchParameters.targetView);
    console.noteln("Star Stretch Process Completed!");
  } else {
    Console.warningln("No target view is specified.");
  }
};

StarStretchDialog.prototype.refreshPreview = function() {
  if (StarStretchParameters.showPreview && StarStretchParameters.targetView) {
    let processingWindow = new ImageWindow(
      StarStretchParameters.targetView.image.width,
      StarStretchParameters.targetView.image.height,
      StarStretchParameters.targetView.image.numberOfChannels,
      StarStretchParameters.targetView.image.bitsPerSample,
      StarStretchParameters.targetView.image.isReal,
      StarStretchParameters.targetView.image.isColor
    );

    if (!processingWindow || processingWindow.isNull) {
      console.writeln("Failed to create processing window.");
      return;
    }

    processingWindow.hide();
    processingWindow.mainView.beginProcess();
processingWindow.mainView.image.assign(StarStretchParameters.targetView.image);
    processingWindow.mainView.endProcess();

    applyPixelMath(processingWindow.mainView, StarStretchParameters.amount);

    if (!isGrayscale(processingWindow.mainView)) {
      applyColorSaturation(processingWindow.mainView,
StarStretchParameters.satAmount);
      if (StarStretchParameters.removeGreen) {
        applySCNR(processingWindow.mainView);
      }
    }

    let tempImage = this.createTemporaryImage(processingWindow.mainView.image);
    if (tempImage) {
      this.previewControl.displayImage = tempImage;
      this.previewControl.viewport.update();
    } else {
      console.writeln("Failed to create a temporary image for preview.");
    }

    processingWindow.forceClose();
  }
};

StarStretchDialog.prototype.createTemporaryImage = function(selectedImage) {
  let window = new ImageWindow(
    selectedImage.width,
    selectedImage.height,
    selectedImage.numberOfChannels,
    selectedImage.bitsPerSample,
    selectedImage.isReal,
    selectedImage.isColor
  );
  window.mainView.beginProcess();
  window.mainView.image.assign(selectedImage);
  window.mainView.endProcess();

  var P = new IntegerResample;
  switch (this.zoomComboBox.currentItem) {
    case 0:
      P.zoomFactor = -1;
      break;
    case 1:
      P.zoomFactor = -2;
      break;
    case 2:
      P.zoomFactor = -4;
      break;
    case 3:
      P.zoomFactor = -8;
      break;
    case 4:
      const previewWidth = this.previewControl.width;
      const widthScale = Math.floor(selectedImage.width / previewWidth);
      P.zoomFactor = -Math.max(widthScale, 1);
      break;
    default:
      P.zoomFactor = -2;
      break;
  }

  P.executeOn(window.mainView);

  let resizedImage = new Image(window.mainView.image);

  if (resizedImage.width > 0 && resizedImage.height > 0) {
    this.previewControl.displayImage = resizedImage;
    this.previewControl.doUpdateImage(resizedImage);
    this.previewControl.initScrollBars();
  } else {
    console.error("Resized image has invalid dimensions.");
  }

  window.forceClose();

  return resizedImage;
};
```

```
// Function to check if the view is grayscale
function isGrayscale(view) {
  return !view.image.isColor;
}

// Modify the main execution logic
function main() {
  // hide the console, we don't need it
  Console.show();
Console.criticalln("   ____    __  _   ___       __        \n  / __/__ / /_(_) / _ | ___ / /_______ ");
Console.warningln(" _\\ \\/ -_) __/ / / __ |(_-</ __/ __/ _ \\ \n/___/\\__/\\__/_/ /_/ |_/__/\\__/_/  \\___/ \n                                        ");
  // script should not run in global mode
  if (Parameters.isGlobalTarget) {
    Console.criticalln("Star Stretch could not run in global context.");
    return;
  }

  // direct context, create and show the dialog
  let dialog = new StarStretchDialog();
  dialog.execute();
}

main();
```