# ADDED BITS FOR THE SARNDBOX

The River Wey Trust SARndbox was built in 2017 as a fully portable unit destined for visits to schools, days out at events, and generally to get around places without too much problem - The worst item is of course lugging the sand, and clearing it up afterwards!

It is a fantastic learning resource, but realistically building one for a school would mean it was used relatively infrequently, either as part of the 'river studies' or for projects in geography or IT - general STEM learning. But storing, maintaining and operations mean that it would be a bit of a faff. I've spoken to quite a few schools, and most often their concern is how to justify the budget for occasional use – a real shame.

Museums and activity centres are perhaps more likely to have a permanent exhibit – but only a few have taken up the challenge, or wish to dedicate exhibition space and management.

Overall therefore the travelling resource is a good option – provided you have an enthusiastic outreach programme to support it!. When dismantled, the Trust's implementation fits in the back of a standard SUV or Estate car, and has put in some miles.

From a slow start in 2017 (it is amazingly hard to get an offer of free resource past the school admin to the teaching staff) we picked up to frequent outings, with 2019 having over 38 events – schools, museums, conferences etc and an estimate of over 2500 hands on users. 2020 started with a couple of schools and the Museum of London half term exhibitions… and visits were looking busier - until the world stopped.

A final flourish though for COVID year - The Royal Institution borrowed our sandbox for their Christmas Lectures, broadcast on the BBC, and now on iPlayer and YouTube – Good PR for the Trust!


Set up to go: Museum of London, Feb 2020

The Trust's version of the SARndbox has a number of 'added features' culled from trawling the forum, and with some home written scripting and configs. Perhaps not always elegant, and

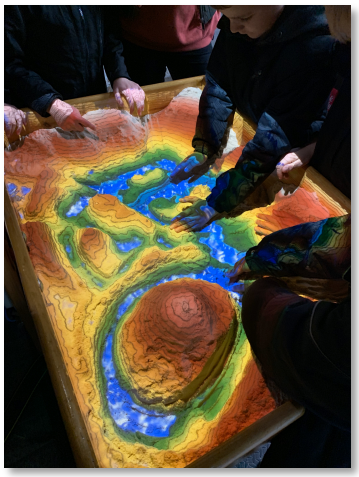
probably inefficient for those more skilled in Linux etc… but for the sake of sharing and supporting Oliver Kraylos' stunning generous gift of the original code and concept, these notes describe what we currently have.

In operation the Trust's SARndbox has four control buttons implemented using a USB encoder (See Appendix lifted directly from the Forum) and a wireless keyboard and trackpad for 'operator' use. Mostly the latter is used in demonstration mode, but sometimes you need 'override' capability when the class gets over excited!

The Trust's website has a few Lesson Plan outlines, and more will be added as they are developed and refined.

ALL THE NOTES HERE WERE WRITTEN WITH VRUI 5.2 AND SARNDBOX 2.7.

ANY UPDATED VERSIONS OF THOSE WILL NEED UPDATES TO THE SCRIPTS AND FILE LOCATIONS ACCORDINGLY.



## INTERESTING PLACES WE VISITED IN THE PAST 12 MONTHS

Several stables and barns, Aldershot Military Museum, The Museum of London, Arthur Conan Doyle's attic, Groundswell Agricultural Conference, The Wey Festival, The Gilbert White Festival, an artist's studio - and a large number of schools… and the Royal Institution …

## TABLE OF CONTENTS

## CONFIGURATION FILES AND SCRIPTS

The River Wey Trust installation has a number of scripted options and uses keyboard assignments and USB buttons to implement a number of effects.

### KEYBOARD

1 Flood

2 Drain

3 Load a DEM

4 Load a different DEM

5 Local Water

6 Local Drain/Dry

7 Sea Level Down in 6 steps

8 Sea Level Up in 6 steps

9 Effect Cycle : Water > Lava > Snow > Pollution > Ice > Toxic > Nasty > …

0 Wiggle Screenshot – takes image of the main display

P SCROT Screenshot – take image of the window in focus

X Effect on/off toggle - reset effect to water and prevent cycle scripts working ( stops 'enthusiastic' users from mashing the buttons to lava when you're trying to talk about water!)

### USB BUTTONS

1 Flood
2 Drain
3 Effect Cycle : Water > Lava > Snow > … (limited to three for user actions)
4 Wiggle Screenshot

**Notes on the use of the effects…**

The USB buttons allow users to manage the display with general effects

The Keyboard allows greater control and allows use in the lesson plans to lead the effects and control use of the display.

The cycled effects (on both Button and Keyboard) are scripted to change the colour of the effect, and also to manage the attenuation and speed of the effect flow behaviour.

The Scripts which are used to implement the change of water effect in the Shader are in a separate document.

The DEM loads are used to add topographical options for use in some of the lesson plans. Where they are not in use, selecting the option removes the colour map from the main display, allowing a further discussion of contours.

The Screenshot options are used in lesson plans to allow comparison records of built landscape against either a target map or a drawing/image.

Lesson plans referenced here are on the website at https://www.riverweytrust.org.uk/resources/

## RUNNING THE SARNDBOX

To enable these to be loaded at start, the script used to load the application is

```
##Setup with SARndbox.fifo pipe for water speeds and sea level changes

# Standard Setup
sh ~/scrip/setweather2.sh #sort the weather to be rain
cd ~/src/SARndbox-2.7
./bin/SARndbox -uhm -fpv -ws 0.85 30 -wi 1 -LoadInputGraph
~/scrip/SARstandard.inputgraph -cp SARndbox.fifo
```

The components called by that script follow:

## CREATING A SCRIPT DIRECTORY

In order to manage the additional scripts and to make any upgrades simpler… suggest that you keep all in a discrete directory

```
mkdir ~/scrip
```

The following are the scripts in my installation, several of these will have an 'echo to screen' component, which will only show if they are executed directly from a terminal. They will not display if the application is started from scripts without a terminal view.

## INPUTGRAPH

This config file is achieved using a VRUI Save Inputgraph option. That has been renamed as `SARstandard.inputgraph` and held in the SCRIP directory. Each section is loading an assignment to key, button, or mouse and loads the 'standard' environment for my installation.

## CAPTURE SCREEN

Using the SCROT utility (install separately) to capture the screen in focus. This facility differs from the WIGGGLE options also used, by allowing the window to be captured to be selected. The Wiggle option is configured to capture an image of the main SARndbox output.

This version takes an image and also copies it to a USB drive with a `/sandbox/CAPTURE` directory sometimes used in lessons.

### CAPTURE-SCREEN.SH

```
# capture-screen.sh
#!/bin/bash

cd ~/Pictures
scrot 'SARndbox_%Y-%m-%d_%H-%M-%s.png' -u -s -e'mv $f /media/sandbox/CAPTURE/'
scrot 'SARndbox_%Y-%m-%d_%H-%M-%s.png' -u
cd ~/src/SARndbox-2.7
```

## 'WEATHER' EFFECTS

Scripts which set and cycle through the colour effects. The comment/echo output will not show unless the sandbox has been initiated from a terminal prompt rather than the bash script.

### SETWEATHER2.SH

Used to initialise the effect to 'water'

```
# setweather2.sh
#!/bin/bash

# pick up current state
  weather=$( cat /home/sandbox/scrip/weather_file.tmp )
  echo "Current state is " $weather "Changed to rain";

  sh /home/sandbox/scrip/switch-to-water.sh ;
  echo "rain" > /home/sandbox/scrip/weather_file.tmp
```

### WEATHER.SH

Cycles through three options water/lava/snow used on USB Button 3

```
# weather.sh
#!/bin/bash

# pick up current state
  weather=$( cat /home/sandbox/scrip/weather_file.tmp )
  echo "Initial state is " $weather

# use case statement to set the assignment to the next cycle value
    case $weather in
     "rain") echo "Changed rain to lava";
            sh /home/sandbox/scrip/switch-to-lava.sh;
            echo "lava" > /home/sandbox/scrip/weather_file.tmp
            ;;
     "lava") echo "Changed lava to snow";
            sh /home/sandbox/scrip/switch-to-snow.sh ;
            echo "snow" > /home/sandbox/scrip/weather_file.tmp
            ;;
     "snow") echo "Changed snow to rain";
            sh /home/sandbox/scrip/switch-to-water.sh ;
            echo "rain" > /home/sandbox/scrip/weather_file.tmp
            ;;
    esac
```

### WEATHER2.SH

Cycles through the full set of effects  - used on keyboard key 9

```
# weather2.sh
#!/bin/bash

# pick up current state
  weather=$( cat /home/sandbox/scrip/weather_file.tmp )
  echo "Current state is " $weather


# use case statement to set the assignment to the next cycle value
    case $weather in
     "rain") echo "Changed rain to lava";
            sh /home/sandbox/scrip/switch-to-lava.sh;
            echo "lava" > /home/sandbox/scrip/weather_file.tmp
            ;;
     "lava") echo "Changed lava to snow";
            sh /home/sandbox/scrip/switch-to-snow.sh ;
            echo "snow" > /home/sandbox/scrip/weather_file.tmp
```

```
            ;;
     "snow") echo "Changed snow to pol";
            sh /home/sandbox/scrip/switch-to-pol.sh ;
            echo "pol" > /home/sandbox/scrip/weather_file.tmp
            ;;
     "pol") echo "Changed polution to ice";
            sh /home/sandbox/scrip/switch-to-ice.sh ;
            echo "ice" > /home/sandbox/scrip/weather_file.tmp
            ;;
     "ice") echo "Changed ice to toxic";
            sh /home/sandbox/scrip/switch-to-toxic.sh ;
            echo "toxic" > /home/sandbox/scrip/weather_file.tmp
            ;;
     "toxic") echo "Changed toxic to nasty";
            sh /home/sandbox/scrip/switch-to-nasty.sh ;
            echo "nasty" > /home/sandbox/scrip/weather_file.tmp
            ;;
     "nasty") echo "Changed nasty to rain";
            sh /home/sandbox/scrip/switch-to-water.sh ;
            echo "rain" > /home/sandbox/scrip/weather_file.tmp
            ;;

    esac
```

### WEATHERHOLD.SH

Assign to key X as a Toggle switch used to pick up the current state and set the effect to water, preventing users interrupting a session by selecting a new effect. This could simply apply rain without the case statement, but written with that and comments just to demonstrate.

```
# weatherHOLD.sh
#!/bin/bash

# pick up current state
  weather=$( cat /home/sandbox/scrip/weather_file.tmp )
  echo "Current state is " $weather


# use case statement to set the assignment to the next cycle value
    case $weather in
     "hold")  echo "Changed hold to rain to allow cycle sequence to restart";
              sh /home/sandbox/scrip/switch-to-water.sh ;
              echo "rain" > /home/sandbox/scrip/weather_file.tmp
              ;;

     "rain")  echo "Changed state to hold and apply water";
              sh /home/sandbox/scrip/switch-to-water.sh ;
```

```
            echo "hold" > /home/sandbox/scrip/weather_file.tmp
            ;;

    "lava")  echo "Changed state to hold and apply water";
            sh /home/sandbox/scrip/switch-to-water.sh ;
            echo "hold" > /home/sandbox/scrip/weather_file.tmp
            ;;

    "snow")  echo "Changed state to hold and apply water";
            sh /home/sandbox/scrip/switch-to-water.sh ;
            echo "hold" > /home/sandbox/scrip/weather_file.tmp
            ;;

     "pol")  echo "Changed state to hold and apply water";
            sh /home/sandbox/scrip/switch-to-water.sh ;
            echo "hold" > /home/sandbox/scrip/weather_file.tmp
            ;;

    "ice")   echo "Changed state to hold and apply water";
            sh /home/sandbox/scrip/switch-to-water.sh ;
            echo "hold" > /home/sandbox/scrip/weather_file.tmp
            ;;

    "toxic") echo "Changed state to hold and apply water";
            sh /home/sandbox/scrip/switch-to-water.sh ;
            echo "hold" > /home/sandbox/scrip/weather_file.tmp
            ;;

    "nasty") echo "Changed state to hold and apply water";
            sh /home/sandbox/scrip/switch-to-water.sh ;
            echo "hold" > /home/sandbox/scrip/weather_file.tmp
            ;;
    esac
```

## SHADER EFFECT SCRIPTS

These scripts are tied into the Cycle commands - Keyboad Key 9 and Button 3. Note that those invoked for Water, Lava, and Snow have specific configurations for speed and attenuation using the Named Pipe - see below for details.
All other effects inherit the settings for Snow.

### WATER SPEED AND ATTENUATION

Once the sandbox is running, you can control it by writing commands to the named pipe. These commands are included in the shader effect scripts. The command is:

```
$ echo "waterAttenuation 0.99" > SARndbox.fifo
```

#### SWITCH-TO-WATER.SH

```
##Select Water Colour shader
cp ~/scrip/SurfaceAddWaterColor-Water.fs ~/src/SARndbox-2.7/share/SARndbox-
2.7/Shaders/SurfaceAddWaterColor.fs
## water attenuation changes pipe for water to 0.0078125
echo "waterAttenuation 0.0078125" > SARndbox.fifo
## water speed changes pipe for water to 0.85
echo "waterSpeed 0.85" > SARndbox.fifo
```

#### SWITCH-TO-LAVA.SH

```
## Load Lava Colour Map
cp ~/scrip/SurfaceAddWaterColor-Lava.fs ~/src/SARndbox-2.7/share/SARndbox-
2.7/Shaders/SurfaceAddWaterColor.fs
## water attenuation changes pipe for LAVA to 0.99
echo "waterAttenuation 0.99" > SARndbox.fifo
## water speed changes pipe for LAVA to 0.40
echo "waterSpeed 0.40" > SARndbox.fifo
```

#### SWITCH-TO-SNOW.SH

```
##Select Water Colour shader
cp ~/scrip/SurfaceAddWaterColor-Snow.fs ~/src/SARndbox-2.7/share/SARndbox-
2.7/Shaders/SurfaceAddWaterColor.fs
## water attenuation changes pipe for SNOW to 0.12
echo "waterAttenuation 0.12" > SARndbox.fifo
## water speed changes pipe for SNOW to 0.039
echo "waterSpeed 0.039" > SARndbox.fifo
```

The next set of effects have no speed or attenuation changes, and simply inherit the existing settings by default - normally 'Snow'.

### SWITCH-TO-ICE.SH

```
cp ~/scrip/SurfaceAddWaterColor-SparklyIce.fs ~/src/SARndbox-2.7/share/SARndbox-2.7/Shaders/SurfaceAddWaterColor.fs
```

### SWITCH-TO-TOXIC.SH

```
cp ~/scrip/SurfaceAddWaterColor-ToxicDeath.fs ~/src/SARndbox-2.7/share/SARndbox-2.7/Shaders/SurfaceAddWaterColor.fs
```

### SWITCH-TO-POL.SH

```
cp ~/scrip/SurfaceAddWaterColor-PollutedWater.fs ~/src/SARndbox-2.7/share/SARndbox-2.7/Shaders/SurfaceAddWaterColor.fs
```

### SWITCH-TO-NASTY.SH

```
cp ~/scrip/SurfaceAddWaterColor-ToxicWaste.fs ~/src/SARndbox-2.7/share/SARndbox-2.7/Shaders/SurfaceAddWaterColor.fs
```

## NAMED PIPE

The three main scripts use a Linux NAMED PIPE to send instructions to manage the speed and attenuation of the flow effects..

To set up the pipe (SARndbox.fifo) you need to issue a one-time command to establish the pipe:

```
cd ~/src/SARndbox-2.7
mkfifo SARndbox.fifo
```

Then run SARndbox with the additional -cp <pipe name> command line argument below - note this is included in the example application startup script above:

```
$ cd ~/src/SARndbox-2.7
$ ./bin/SARndbox <usual parameters> -cp SARndbox.fifo
```

## HEIGHT AND SEA LEVEL VARIATIONS

From Oliver's notes:

*Contour lines are generated by the surface rendering fragment shader. The exact code is in Shaders/SurfaceAddContourLines.fs. Basically, the shader detects whether a given screen pixel is intersected by at least one contour line, and if so, colours that pixel black. That's why the contour lines can't easily be labelled.*

*The base plane equation is stored in two classes, for different purposes. The DepthImageRenderer class contains the base plane equation used to generate contour lines and a bathymetry DEM for water simulation, and the ElevationColorMap class contains the base plane equation used to colour-map the surface. Normally those two are the same, but they don't need to be.*

*The function to change the colour mapping base plane dynamically is exposed at the SARndbox's pipe command interface.*
*If you create a named pipe and attach to it with* `./bin/SARndbox ... -cp <pipe name>` *and then write the following to the pipe while the sandbox is running:*

```
echo "heightMapPlane <normal x> <normal y> <normal z> <offset>"
```

*(for example:* `echo "heightMapPlane 0.0 0.0 1.0 -95.0"` *to put the base plane flat at 95cm below the camera)*

*it will immediately change the elevation colour mapping.*

*If you want to change the* ***contour line base elevation*** *at the same time, call depthImageRenderer->setBasePlane() with the same plane equation from the same location in the code (inside Sandbox::frame).*

*In short, insert*

```
/* Override the elevation base plane in the depth image renderer: */
depthImageRenderer->setBasePlane(heightMapPlane);
```

*before line 1152 in Sandbox.cpp.*

## SEA LEVEL 'HEIGHT' CHANGE SCRIPTS

There are now TWO versions for changing heights - the first using the base plane changes suggested above, whilst the second simply changes the 'z' component.

### THE FIRST METHOD

Examples using the named pipe as described above for amending the Height Map Plane to adjust the sea level in 2 or 3 cm increments - this is achieved here with two separate scripts… one to take the sea level down, one to take it up.

These scripts use SARndbox.fifo to amend the configuration 'in flight', and are based on the default plane (-102) in my installation - determined in the BoxLayout.txt config file. (SARndbox-xxx/etc/SARndbox-xxx/Boxlayout.txt)

My implementation assigns the scripts to keys 7 and 8 respectively (*but see the second method which uses different key assignments*)

#### HEIGHTS-DOWN.SH

```
# heights-down.sh
#!/bin/bash

# pick up current state
  currdep=$( cat /home/sandbox/scrip/height_set.temp )
  echo "Initial state is " $currdep

# use case statement to set the assignment to the next cycle value
    case $currdep in
     "-96") echo "Reset Depth to -98";
             echo "heightMapPlane 0.0 0.0 1.0 -98.0" > SARndbox.fifo
             echo "-98" > /home/sandbox/scrip/height_set.temp
            ;;
     "-98") echo "Reset Depth to -100";
             echo "heightMapPlane 0.0 0.0 1.0 -100.0" > SARndbox.fifo
             echo "-100" > /home/sandbox/scrip/height_set.temp
            ;;
     "-100") echo "Reset Depth to -102 DEFAULT";
             echo "heightMapPlane 0.0 0.0 1.0 -102.0" > SARndbox.fifo
             echo "-102" > /home/sandbox/scrip/height_set.temp
            ;;
     "-102") echo "Reset Depth to -105";
             echo "heightMapPlane 0.0 0.0 1.0 -105.0" > SARndbox.fifo
             echo "-105" > /home/sandbox/scrip/height_set.temp
            ;;
     "-105") echo "Reset Depth to -108";
             echo "heightMapPlane 0.0 0.0 1.0 -108.0" > SARndbox.fifo
             echo "-108" > /home/sandbox/scrip/height_set.temp
```

```
            ;;
     "-108") echo "Reset Depth to -110";
             echo "heightMapPlane 0.0 0.0 1.0 -110.0" > SARndbox.fifo
             echo "-110" > /home/sandbox/scrip/height_set.temp
            ;;
     "-110")  echo "Not Going Any Lower!";
            ;;
    " *"  )  echo "Reset to Base Plane default";
             echo "heightMapPlane 0.0 0.0 1.0 -102.0" > SARndbox.fifo
             echo "-102" > /home/sandbox/scrip/height_set.temp
            ;;

    esac
```

## HEIGHTS-UP.SH

```
# heights-up.sh
#!/bin/bash

# pick up current state
  currdep=$( cat /home/sandbox/scrip/height_set.temp )
  echo "Initial state is " $currdep

# use case statement to set the assignment to the next cycle value
    case $currdep in
     "-110") echo "Reset Depth to -108";
             echo "heightMapPlane 0.0 0.0 1.0 -108.0" > SARndbox.fifo
             echo "-108" > /home/sandbox/scrip/height_set.temp
            ;;
     "-108") echo "Reset Depth to -105";
             echo "heightMapPlane 0.0 0.0 1.0 -105.0" > SARndbox.fifo
             echo "-105" > /home/sandbox/scrip/height_set.temp
            ;;
     "-105") echo "Reset Depth to -102 DEFAULT ";
             echo "heightMapPlane 0.0 0.0 1.0 -102.0" > SARndbox.fifo
             echo "-102" > /home/sandbox/scrip/height_set.temp
            ;;
     "-102") echo "Reset Depth to -100";
             echo "heightMapPlane 0.0 0.0 1.0 -100.0" > SARndbox.fifo
             echo "-100" > /home/sandbox/scrip/height_set.temp
            ;;
     "-100") echo "Reset Depth to -98";
             echo "heightMapPlane 0.0 0.0 1.0 -98.0" > SARndbox.fifo
             echo "-98" > /home/sandbox/scrip/height_set.temp
            ;;
     "-98") echo "Reset Depth to -96";
             echo "heightMapPlane 0.0 0.0 1.0 -96.0" > SARndbox.fifo
             echo "-96" > /home/sandbox/scrip/height_set.temp
            ;;
     "-96")  echo "Not Going Any Higher!";
```

```
            ;;
     "*" )   echo "Reset to normal Base Plane";
             echo "heightMapPlane 0.0 0.0 1.0 -102.0" > SARndbox.fifo
             echo "-102" > /home/sandbox/scrip/height_set.temp
            ;;

    esac
```

## THE SECOND METHOD

Possibly more 'elegant' and allowing unlimited changes, this is a much simpler script than the first method, simply amending the 'z' value in single steps of 0.01 and rewriting the heightMapPlane string through the named pipe SARndbox.fifo to amend the configuration 'in flight'.

My implementation assigns the scripts to keys U (for up) and D (for down). I've also assigned key R to a reset function to put the display back to 'normal'.

### HEIGHTS-LOWER.SH

```
# heights-lower.sh
#!/bin/bash

# pick up current state
  currlevel=$( cat /home/sandbox/scrip/height_level.temp )

    echo "Initial state is " $currlevel

        adder=0.01
        currlevel=`echo $currlevel - $adder | bc`
    echo "Decremented Value ;" $currlevel

    echo "$currlevel"  > "/home/sandbox/scrip/height_level.temp"
    echo "heightMapPlane 0.0 0.0 " $currlevel " -102.0" > SARndbox.fifo
```

### HEIGHTS-RAISE.SH

```
# heights-raise.sh
#!/bin/bash

# pick up current state
  currlevel=$( cat /home/sandbox/scrip/height_level.temp )

    echo "Initial state is " $currlevel

        adder=0.01
        currlevel=`echo $currlevel + $adder | bc`
```

```
    echo "Incremented Value ;" $currlevel

    echo "$currlevel"  > "/home/sandbox/scrip/height_level.temp"
    echo "heightMapPlane 0.0 0.0 " $currlevel " -102.0" > SARndbox.fifo
```

### DEFAULT_SEALEVEL_SETTING.SH

```
# default_sealevel_setting.sh
#!/bin/bash
echo "heightMapPlane 0.0 0.0 1.0 -102.0" > SARndbox.fifo # default sea level
setting
echo "-102" > /home/sandbox/scrip/height_set.temp
echo "1.0"  > /home/sandbox/scrip/height_level.temp
echo "Reset Depth Plane to -102  and Z value=1.0 AS DEFAULTS ";
```

## ALTERNATIVE STARTUPS

Make and Edit files to appropriate user and SARndbox version – Whilst such features as Triangulated Water are excellent visualisations, it is sometimes better to keep the monitor views 'in step' – especially if you are using the 'lava/snow/etc' display option features.

The simplest way to do that is to have two discrete startup scripts - one with the Triangulated Water parameter ( -rws ), and one without. Simply clone the RunSARndbox.sh script, and insert the parameter saving as – for example RunSARndbox-TRI.sh

The example scripts below run an initial script to default the water effect to water rather than lava etc, invoke the Sandbox with standard parameters, controlling also the speed of water, defining a second monitor, and applying an inputgraph which is used to define the button and key assignments appropriate to this setup. The last line invokes the Named Pipe to allow in-flight instructions.

```
################################################################################
# 4 Buttons : Flood, Drain, Cycle x3 effects, ScreenCapture
# Keys=
# 1: Flood
# 2: Drain
# 3: DEM
# 4: DEM
```

```
# 5: Local Water
# 6: Local Dry
# 7 Lower Sealevel
# 8 Raise Sealevel
# 9: Cycle all effects
# 0: Wiggle Screenshot
# X: Suspend/Enable effects
# P: SCROT Screen Capture

##Setup with SARndbox.fifo pipe for water speeds and sea level changes

# Standard Setup
sh ~/scrip/setweather2.sh #sort the weather to be rain
cd ~/src/SARndbox-2.7
./bin/SARndbox -uhm -fpv -ws 0.85 30 -wi 1 -LoadInputGraph
~/scrip/SARstandard.inputgraph -cp SARndbox.fifo
```

Alternative…

```
################################################################################
# Sandbox WITH triangulated Water
sh ~/scrip/setweather2.sh #sort the weather to be rain
cd ~/src/SARndbox-2.7
./bin/SARndbox -uhm -fpv -ws 0.85 30 -wi 1 -rws -LoadInputGraph
~/scrip/SARstandard.inputgraph -cp SARndbox.fifo
```

.. and a further Option

The Trust's Kinect/projector setup means that the additional monitor initially shows the view in 'reverse' -i.e. back to front as far as the sandbox is concerned. Rather than rotate the Kinect with all the wires coming out the wrong way and having to reconfigure the whole setup, it was simpler to use the 'Set View option' to rotate the display on the monitor.

This is achieved by using the dialogue to save a view - similar to the process for saving the inputgraph, the loading that view as part of the startup script… which now for the 'standard' startup becomes..

```
##Setup with SARndbox.fifo pipe for water speeds and sea level changes

# Standard Setup
sh ~/scrip/setweather2.sh #sort the weather to be rain
cd ~/src/SARndbox-2.7
./bin/SARndbox -uhm -fpv -ws 0.85 30 -wi 1 -LoadInputGraph
~/scrip/SARstandard.inputgraph -LoadView ~/scrip/Set3D.view -cp SARndbox.fifo
```

## EXAMPLE SARNDBOX.CFG FILE

```
# This file is ~/.config/Vrui-5.2/Applications/SARndbox.cfg
#
section Vrui
    section Desktop
        # Disable the screen saver:
        inhibitScreenSaver true

        section MouseAdapter
            # Hide the mouse cursor after 5 seconds of inactivity:
            mouseIdleTimeout 5.0
        endsection

        # Open a second window:
        windowNames += (Window2)

        section Window
            # Open the window on a specific video output port:
            outputName HDMI-0

            # Force the application's window to full-screen mode:
            windowFullscreen true
        endsection

        section Window2
            viewerName Viewer
            screenName Screen
            windowType Mono

            # Open the window on a specific video output port:
            outputName DP-5

            # Open the window to full-screen mode:
            # windowSize (800, 600)
            # windowFullscreen true
        endsection

        inputDeviceAdapterNames += (HIDAdapter)
        ...
        section HIDAdapter
            inputDeviceAdapterType HID
            inputDeviceNames (USBEncoder)

            section USBEncoder
                name USBEncoder
                deviceVendorProductId 0079:0006
                trackingDeviceName Mouse
            endsection
        endsection
```

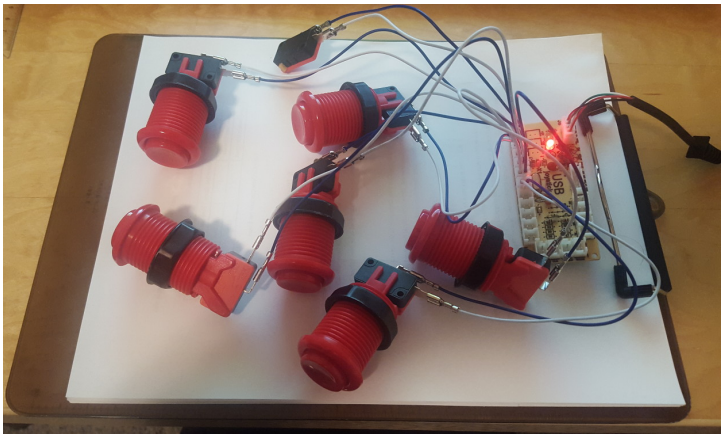
```
        section Tools
            section DefaultTools
                # Bind a global rain/dry tool to the "1" and "2" keys:
                section WaterTool
                    toolClass GlobalWaterTool
                    bindings ((Mouse, 1, 2))
                endsection
            endsection

            toolClassNames += (ScriptExecutorTool)

        endsection
    endsection
```

## THE USB BUTTONS SEGMENT

***These notes are taken directly from Oliver's piece in the Forum.***



Reyann Zero Delay Arcade USB Encoder.

*Most USB buttons emulate keys, and have a bug in key repeat handling that can cause input buffer overflows and AR Sandbox hangs. This encoder directly advertises up to twelve generic HID buttons, which is more stable and also does not cause interference with desktop applications.*

*To assemble the encoder, simply plug the provided USB cable into the matching four-prong plug on the circuit board, and your desired number of button cables into the two-prong button plugs along the opposite long edge of the board. Connect the other ends of the button cables firmly to the terminals on your microswitches; polarity does not matter.*

*Do not use the four two-prong plugs on the short edge; those have special functions and are not sent over the HID protocol. The button plugs along the long row are named "Button0" to "Button11", in increasing order away from the two power and mode indicator LEDs. You can use any subset of button plugs, they don't need to be connected consecutively.*

*After assembling the controller, visually double-checking for short circuits, and plugging the USB cable into your computer, run*

```
lsusb
```

*from a terminal to check that the encoder is detected. It will be listed as:*

```
Bus XXX Device YYY: ID 0079:0006 DragonRise Inc. PC TWIN SHOCK Gamepad
```

*where the bus and device numbers depend on your local setup.*

*Making the encoder usable from the AR Sandbox requires a few simple steps. First, create a device rule to allow non-administrative users to access the button device. From a terminal, run*

```
sudo xed /etc/udev/rules.d/69-USB-encoder.rules
```

*Into the empty file, insert exactly the following line:*

```
SUBSYSTEMS=="usb", ATTRS{idVendor}=="0079", ATTRS{idProduct}=="0006", TAG+="uaccess"
```

*then save the file and exit the text editor. Unplug and re-plug the encoder's USB cable to activate the new rule.*

*Next, open the AR Sandbox's patch configuration file …*

```
xed ~\.config\Vrui-5.2\Applications\SARndbox.cfg
```

*and insert the following lines into the "Desktop" section:*

```
section Vrui
    section Desktop
        ...
        inputDeviceAdapterNames += (HIDAdapter)
        ...
        section HIDAdapter
            inputDeviceAdapterType HID
            inputDeviceNames (USBEncoder)

            section USBEncoder
                name USBEncoder
                deviceVendorProductId 0079:0006
                trackingDeviceName Mouse
            endsection
        endsection
        ...
    endsection
endsection
```

*Save the new or edited file, and run the AR Sandbox as usual. If you now press and hold any of the connected buttons, the usual Vrui tool selection menu pops up. Move the mouse to select a tool, and release the pressed button. Then assign additional buttons as usual.*

*To make the new tool assignments permanent, save the input graph via the main menu's "Vrui System"->"Devices"->"Save Input Graph" entry, and optionally add the new bindings to the AR Sandbox's patch configuration file as explained in the forum post linked above. The device name to use for tool bindings is "USBEncoder", and the button names are "Button0" to "Button11".*

*One side effect of using a generic HID button interface is that it is no longer possible to bind desktop shortcuts to the buttons to execute external scripts, such as switching between water and lava rendering modes or changing colour maps. To fix this, and to centralize and simplify script management, Oliver added a "Script Executor" tool class to Vrui that can execute scripts from within a Vrui application, bypassing the desktop environment's mechanism.*

*Running the AR Sandbox as normal, press some button and select a "Script Executor" tool from the "Utility" submenu of the tool selection menu. A file selection box will pop up where you can select the script you want to execute*

*Alternatively if you wish, add tool binding sections for script tools directly to the AR Sandbox's patch configuration file. Use the tool class name "ScriptExecutorTool" and specify the name of the script to be executed by inserting the following tag into the tool binding section:*

```
executablePathName /path/to/and/name/of/your/script
```

*It is also possible to pass command line arguments to scripts, but this can only be done by editing a configuration file or an input graph file. In the appropriate tool binding section, insert a tag*

```
arguments ("argument 1", "argument 2", ..., "argument n")
```

*where each argument is enclosed in double quotes.*

## ADDENDUM


The Buttons provided as a set with the USB encoder suffer from 'sand jamming' when enthusiastic and sandy fingers manage to get sand into the mechanism, making them stick and repeat constantly… As an alternative I use some more robust buttons - also from Amazon "Vandal Resistant 22mm Stainless Steel Momentary Push Button Switch 2A SPST" https://www.amazon.co.uk/gp/product/B07BR7WD8Y/ref=ppx_yo_dt_b_asin_title_o04_s00?ie=UTF8&psc=1.