

Final Report for NRL

Development of Display Environment for LASCO, EIT and SECCHI

Prepared by
Tae Jin Kim, Cole Krumbholz, Jason Leigh
 Electronic Visualization Laboratory, University of Illinois at Chicago
 July 10, 2007

● Background

Researchers in the [Solar Physics Branch](#) at [Naval Research Laboratory \(NRL\)](#) have been involved in observational and theoretical studies of the solar atmosphere since the early years of the space age. They are viewing impressive solar disturbances whose depth and violent nature are now clearly visible in the first true stereoscopic images ever captured of the Sun. These new views, recently released by NASA, are providing scientists with unprecedented insight into solar physics and the violent solar weather events that can bombard Earth's magnetosphere with particles and affect systems ranging from weather to our electrical grids.

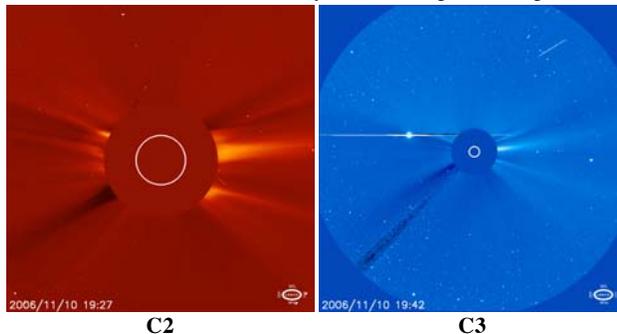
[Electronic Visualization Laboratory \(EVL\)](#) has helped establish a solar imagery display environment, two EVL-developed display systems capable of viewing and managing files on the scale of thousands of pixels per square inch: a 9-panel tiled LCD wall ideal for viewing high-resolution 2D imagery, and an ImmersaDesk4 for viewing high-resolution 3D imagery.

The tiled display consists of 9 cinema LCD displays with an overall resolution of approximately 8k by 5k pixels and will be used to work with multiple HD quality video feeds, enabling multi-spectral analysis of synchronized image streams from sensors on each satellite. The ImmersaDesk4, invented at the University of Illinois at Chicago (UIC) Electronic Visualization Laboratory (EVL), is a tracked, 4-million-pixel display system driven by a 64-bit graphics workstation. Its compact workstation design is comprised of two 30-inch Apple LCD monitors mounted with quarter-wave plates and bisected by a half-silvered mirror enable circular polarization. It permits scientists to visualize the stereo imagery in high resolution 3D. Multiple users can view the head-tracked 3D scene using lightweight polarized glasses.

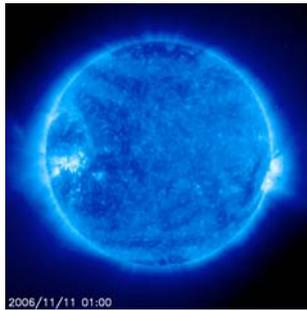
Solar physicists at NRL hope that this new technology will allow them to better visualize the three dimensional structure of coronal mass ejections and evaluate the impact these events have on the earth's atmosphere.

What are LASCO, EIT and SECCHI?

- LASCO (the Large Angle and Spectrometric Coronagraph) is a set of three "coronagraph" telescopes on-board the SOHO satellite. A coronagraph is a special type of telescope that uses a solid disk ("occultor" or "occluding disk") to actually cover up the Sun itself, completely blocking direct sunlight, and allowing us to see the atmosphere around the outside of the Sun. LASCO comprises of three telescopes (C1, C2 and C3), each of which looks at an increasingly large area surrounding the Sun. For the first year-and-a-half of the SOHO mission, all three instruments worked perfectly. However, in 1998 SOHO was accidentally "lost" in space after it received a bad command. The entire spacecraft lost power and essentially froze solid for several weeks. Eventually -- miraculously! -- the SOHO team were able to relocate the spacecraft, regain control and slowly power-up and thaw out the instruments. Sadly, the LASCO C1 camera was lost as a result of this but the rest of spacecraft came through almost completely unscathed! Eight years later -- and over ten years since launch -- LASCO C2 and C3 (and most of the rest of SOHO!) continue to work extremely well, sending back images and data on a daily basis.

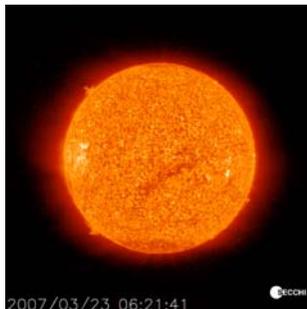


- EIT (the Extreme ultraviolet Imaging Telescope) is another of the instruments on-board SOHO. Unlike LASCO it is not a coronagraph, but instead takes direct images of the Sun using different filters that allow us to see different layers of the Sun's outer atmosphere. An example of an EIT image can be seen opposite. Although they are completely separate instruments, LASCO and EIT share a lot of the electronics on the SOHO spacecraft and the NRL Solar Physics Branch are responsible for support for the camera and electronics of EIT, as well as being the principal investigation team for the LASCO instrument.



EIT

- SECCHI (Sun Earth Connection Coronal and Heliospheric Investigation) is a suite of 5 scientific telescopes SCIP, HI and SEB that will observe the solar corona and inner heliosphere from the surface of the Sun to the orbit of Earth. These unique observations will be made in stereo from NASA's Solar Terrestrial Relations Observatory STEREO. The STEREO mission was successfully launched at 8:52pm EDT on October 25, 2006 from Cape Canaveral in Florida atop a Delta II rocket.



SECCHI

Dataset for LASCO, EIT and SECCHI

- The number of images for each sensor is currently between about 10 and about 790. It means that whole images for a specified sensor are buffered, and then displayed sequentially on the videowall
- each image is up to 2K by 2K pixels

● Download and Compile

The source code for this software is managed by subversion software, a revision control system.

- svn root : <https://svn.sourceforge.net/svnroot/gpip/branches/apps/viewer>

svn co https://svn.sourceforge.net/svnroot/gpip/branches/apps/viewer

The directories of source code

l---- config : the lua config files used to configure the viewer for your particular tiled display

l---- gpip : the GPIIP library

l---- imgbuffer : manage image files

l---- scripts : the control scripts that determine the viewer's behavior.

l---- shaders : the programmable shaders used by GPIIP

l---- utils : font, and common list

l---- viewer : contains main logic (solarvision.c/h) and the lua scripting engine (config.c/h)

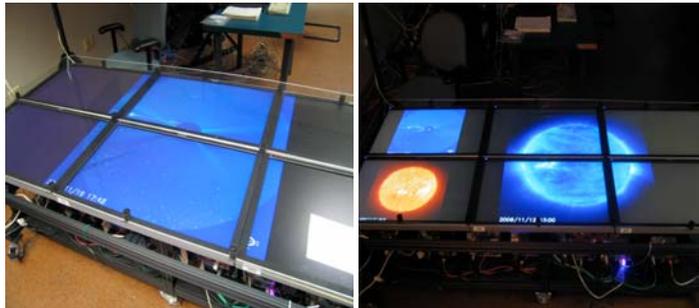
l---- window

- current installed softwares and libraries

mpich	v1	http://www-unix.mcs.anl.gov/mpi/mpich1/index.htm
sdl	v1.2.9	http://www.libsdl.org
lua	v5.0.2	http://www.lua.org
freetype	v.8.3	http://www.freetype.org
cfitsio	v3.006	http://heasarc.gsfc.nasa.gov/docs/software/fitsio/fitsio.html
libpng	v3.1.2.8	
libjpeg	v6.0.1	
For GUI Server		
luasocket	v2.0.1	http://www.cs.princeton.edu/~diego/professional/luasocket/home.html

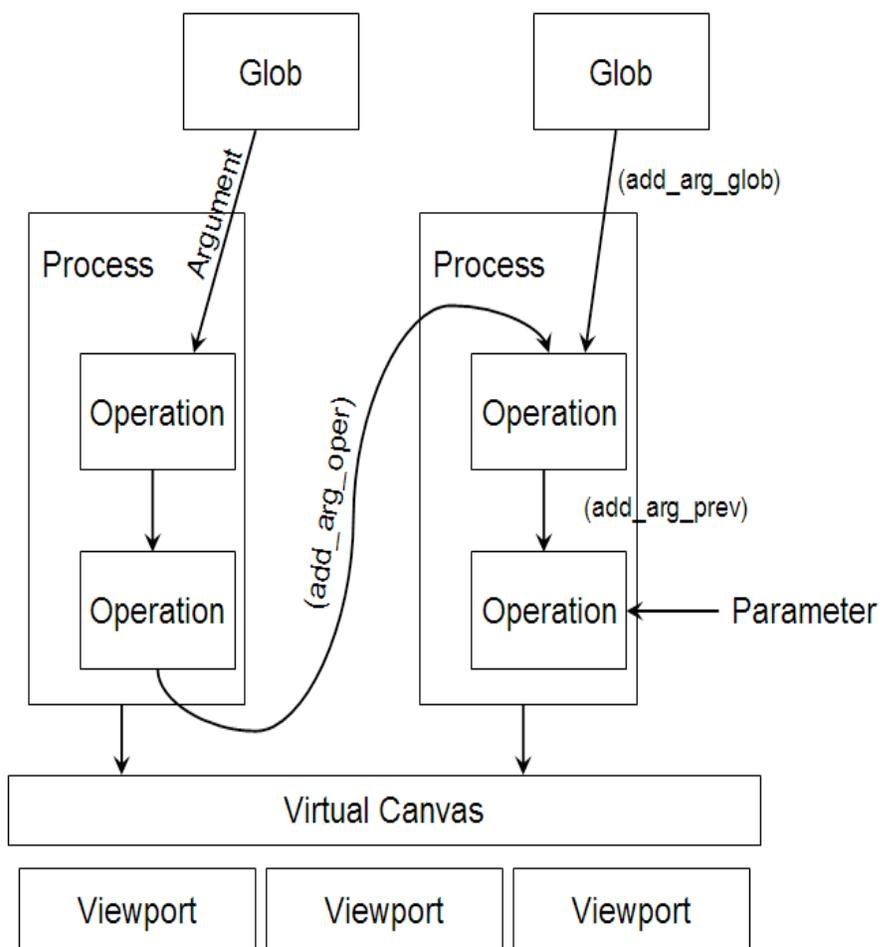
For GUI Client		
python	v2.4.2	http://www.python.org/index.php
wxPython	v2.8	http://www.wxpython.org/index.php

- compile
 - cd \$SOURCE_ROOT/viewer
 - make
- run
 - mpirun -np *node_number* -machinefile *node_file* *executable_file* -f *configuration_script* -f *application_script* -d *data_directory* -t *data_type*
 - option -f
load any lua script, configuration_script and application_script is written by lua script.
 - option -d
data type list
general image : -d **general**
lasco image : -d **lasco**
secchi image : -d **secchi**
 - [example] mpirun -np 4 -machinefile ./config/teiburu.dat ./viewer/viewer -f ./config/teiburu.lua -f singlemov.lua -d /data/lasco/c3 -t lasco



● software overview

- The goal is to pretend that the tiled display is actually one large seamless display. This is done by creating one large virtual canvas, drawing images and text onto that canvas, and rendering a piece of the canvas into each monitor's viewport. The configuration file written by Lua script is used to define the size of the size of the virtual canvas and the position/size of each monitor's viewport. During each rendering pass the virtual canvas is updated, and then for each display the canvas is drawn, but cropped according to that display's viewport. This is a typical approach for rendering multiple views using a graphics library like OpenGL.
- This software design is grouped into four major components.
 - GPIP : Graphics Processor enhanced Image Processing library using the GPU and programmable shaders to speed up image processing tasks.
 - imgbuffer : a component for managing a large set of image files and paging images into memory for video playback.
 - A configuration and scripting front end that uses the Lua scripting language.
 - viewer : the main logic that puts these pieces together.
- About how GPIP works:
GPIP manages image processing pipelines (procs) composed of operations performed on global image buffers (globs). Generally when using gpip you define a proc, add operations, add arguments to those operations using globs, and then arrange the output on the screen. The operations are defined inside shaders that use the GLSL shading language to perform the appropriate image processing logic. Though the library provides a mechanism for fast image processing, it requires that the user understand how to write GLSL shaders. For example, to set up a box filter on an image, first create a GLSL shader (just a fragment shader) that takes a texture input image, and then for each pixel writes out the average value for that region of adjacent pixels. Then in GPIP create a proc, and add that shader as an operation. Then define the arguments for that operation, in this case there is a single argument, which is an input image, or glob. If we wanted to add additional operations, those operations could use the previous result as input, or other proc results, or more global image buffers (globs).



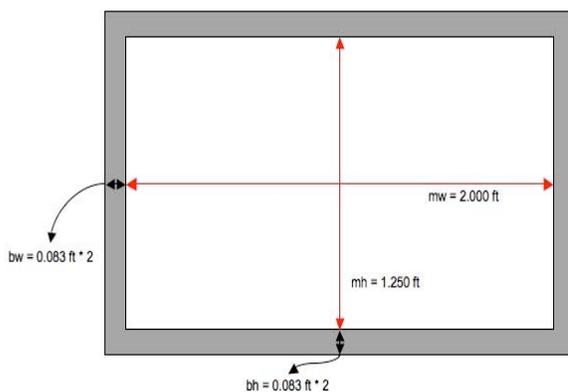
how these pieces work together

● Configuration

The configuration is written in lua script, used to configure the viewer for your particular tiled display.

- add host
add_host(hostname, width, height, fullscreen)
 adds a new computational node. Hosts can be specified by hostname or hostname:display. Multiple hosts can exist on a single physical computer if that computer is configured with multiple displays.
- add tile
add_tile(hostid, vx, vy, vw, vh, va, wx, wy, ww, wh)
 adds a new monitor. Multiple tiles can exist on a single host if that host's display is spread across more than one physical monitor. The view of the tile into the virtual canvas is specified by vx, vy, vw, vh. The orientation of the tile is specified by va. The actual window that the tile occupies in host display coordinates is specified by wx, wy, ww, wh.

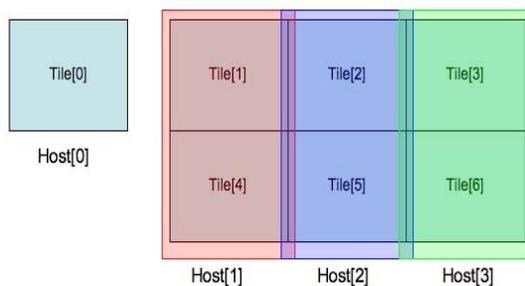
Configuration of videowall at EVL



../config/teiburu.conf

```
mw = 2.000          -- monitor width
mh = 1.250          -- monitor height
bw = 0.166          -- border width
bh = 0.166          -- border height
```

•



../config/teiburu.conf

```
host[0] = S.add_host("teiburu",      1280, 1024, 0)
host[1] = S.add_host("teiburu1-10",  2560, 3200, 1)
host[2] = S.add_host("teiburu2-10",  2560, 3200, 1)
host[3] = S.add_host("teiburu3-10",  2560, 3200, 1)

-- master (host[0])
tile[0] = S.add_tile(host[0], 0,      0,      dw, dh, 0,  0, 0,  1280, 1024)
-- the first row
tile[1] = S.add_tile(host[1], 0,      0,      mw, mh, 180, 0, 1600, 2560, 1600)
tile[2] = S.add_tile(host[2], mw+bw,  0,      mw, mh, 180, 0, 1600, 2560, 1600)
tile[3] = S.add_tile(host[3], 2*mw+2*bw, 0,      mw, mh, 180, 0, 1600, 2560, 1600)
-- the second row
tile[4] = S.add_tile(host[1], 0,      mh+bh, mw, mh, 180, 0, 0,      2560, 1600)
tile[5] = S.add_tile(host[2], mw+bw,  mh+bh, mw, mh, 180, 0, 0,      2560, 1600)
tile[6] = S.add_tile(host[3], 2*mw+2*bw, mh+bh, mw, mh, 180, 0, 0,      2560, 1600)
```

•

Application Script

Application script is used for displaying images of LASCO, EIT and SECCHI. It specifies the directory and type of those images, the position of a video on virtual canvas.

function	return	description
----------	--------	-------------

add_sequence()	sequenceid	creates a new sequenced set of images. The sequence may have more than one set of images, but will only have one timeline. The images are time-stamped, and when multiple image sets are attached to a particular sequence, they will be synchronized to the same timeline
[example] sequence = S.add_sequence();		
add_stream(sequenceid, data_directory, data_type)	streamid	creates a new set of time-stamped images. All files in the given directory path will be added to the stream. Each file in the directory should be a secchi or lasco image with the appropriate time- stamp embedded in the filename
[example] sensor.stream = S.add_stream(sequence, sensor.dir, dir_type)		
add_video(streamid)	videoid	add_video() creates a video from a stream of images. The stream argument must be provided by a previous call to add_stream()
[example] sensor.vid = S.add_video(sensor.stream)		
load_sequence(hostid, sequenceid)		prepares a sequence for viewing on a particular host. Currently only one sequence can be loaded on a given host. Images from sequences not loaded on a host will not be visible to that host's tiles.
[example] S.load_sequence(host[i], sequence)		
set_window(videoid, x, y, width, height, angle);		maps the video to a rectangle within the virtual canvas. The rectangles origin is specified by wx, wy and its width and height are specified by ww, wh. The rectangle will be rotated about its origin by wa degrees
[example] S.set_window(sensor.vid, x - w/2, y - w/2, w, w, 0);		
set_diff_ramp(ramp_file)		set_diff_ramp() specifies the image file that contains the colormap to be applied to all running difference images. It should be a 256 pixel wide image with each pixel along the x axis providing a 3 channel color value for greyscale level corresponding to the pixels x coordinate.
[example] S.set_diff_ramp("../ramps/cole_bwmid.jpg");		
add_diff (videoid);		creates a running difference video from another video. The video argument must be provided by a previous call to add_video()
[example] sensor.dif = S.add_diff (sensor.vid);		
add_text(string, x, y)		create a string and places it at a desired position 
[example] S.add_text("This is sun", 2560 + 1000, 1600 + 1000)		
set_hide_video(videoid, 0 or 1);		hides video, set by the function, add_video
[example] S.set_hide_video(sensor.vid, 0);		
set_targ_fps(fps_value, movie_direction);		sets the desired frames per second and movie's direction (forward or backward)

[example] S.set_targ_fps(targ_fps, dir);		
set_targ_colormap(offset_value, scale_factor)		sets or adjusts colormap's offset and scale factor
[example] S.set_targ_colormap(cmap_offset, cmap_scalef);		
set_mouse(mouse_x, mouse_y, 0 or 1)		sets the desired position of virtual mouse on the virtual canvas
[example] S.set_mouse(mouse_x, mouse_y, 1);		

● Stereo view - ImmersaDesk4

The [ImmersaDesk4 \(IDesk4\)](#) is built from 2 Apple 30" 2560 x 1600 LCD panels mounted with quarter-wave plates in front of the LCD panels to achieve circular polarization.

x-window configuration	Configuring TwinView	[example] xorg.conf
configuration script	<ul style="list-style-type: none"> • set two tiles, using the function "add_tile" • adjust the offset for left and right eye, respectively, using the function "set_tile_view_offset" set_tile_view_offset(tileid, x_offset, y_offset)	[example] argos_idesk4.conf
run	It is the same way as the single view. The only difference is that "mpirun" is not needed to run, because the host is only one.	[example] <pre>../viewer/viewer -f ../config/argos_idesk4.lua -f ../scripts/singlemov.lua</pre>



● GPU shader for colormap

The videowall passes the colormap and the index of color into a GPU, and the vertex shader of the GPU specifies the value of color for each vertex.

download : [colormap.frag](#)

```
#extension GL_ARB_texture_rectangle : enable

/* passing from the videowall application with OpenGL */

uniform sampler2DRect image; // image texture location
uniform sampler2DRect colormap; // colormap texture location

uniform float cmap_offset; // offset of colormap
uniform float cmap_scalef; // scale factor of colormap

void main(void)
{
    // do a lookup into image texture
    float color = texture2DRect(image, gl_FragCoord.st).r;

    // re-calculate the color's index for colormap
    vec2 map = vec2(color * 255.0 * cmap_scalef + 0.5 + cmap_offset, 0.0);

    // do a lookup into colormap texture
    vec4 mapColor = texture2DRect(colormap, map);

    // assign the color value for a vertex
    gl_FragColor = mapColor;
}
```

● The description of control key ("[singlemov script](#)")

movie control	keyboard	: description
	up arrow	: increase fps
	down arrow	: decrease fps

	right arrow : moving forward left arrow : moving backward spacebar : pause moving
colormap control	keyboard : description key "c" : increase a offset key "z" : decrease a offset key "s" : increase a scale factor key "x" : decrease a scale factor
virtual mouse control	keyboard : description key "j" : left key "l" : right key "i" : up key "k" : down key "p" : increase a movement interval key "u" : decrease a movement interval : display the pixel value on current mouse position key "m"



● GUI server and client

GUI server

- Compile : We don't need to do anything special
- How to run the videowall with GUI server
 - mpirun -np node_number -machinefile node_file executable_file -f configuration_script -f finding_luasocket_script -f guiserver_script -f application_script -d data_directory -t data_type
 - [Example] mpirun -np 4 -machinefile ../config/teiburu.dat ../viewer/viewer -f ../config/teiburu.lua -f /usr/local/share/lua/5.1/compat-5.1.lua -f [guiserv.lua](#) -f singlemov.lua -d /data/lasco/c3 -t lasco

GUI client

- How to run : python [nrl_client.py](#)
- currently, a gui client and server are using a "3333" port number
- connect / disconnect

Server:

- control colormap
 - scale factor, "-" / "+" button : adjust a scale factor
 - offset, "<<" / ">>" button : adjust a offset

Colormap Control

scale factor :

offset :

- control playing movie
 - speed control, slide bar : adjust a speed
 - pause button : pause playing
 - direction, "<<"/ ">>" button : backward / forward playing

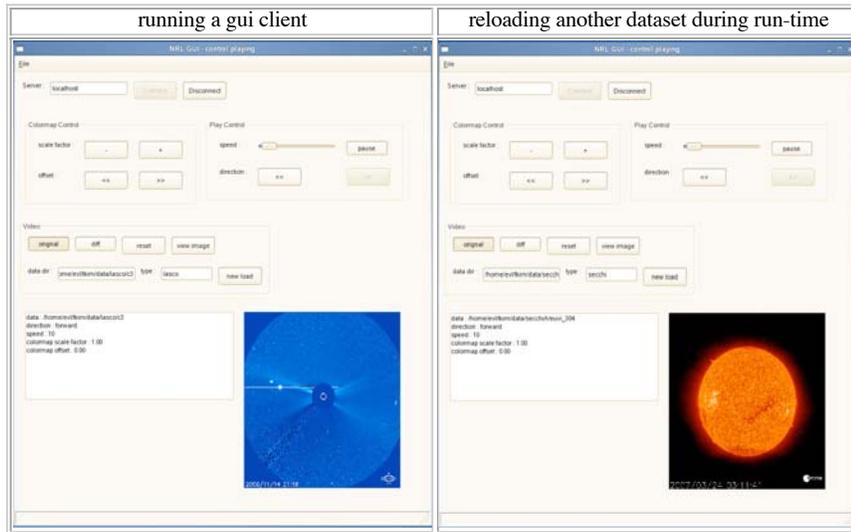
Play Control

speed :

direction :

- control video
 - original button : display a original video

- diff button : display a video with difference mode
- reset button : reload a video and reset
- view image button : get current displaying image (at running only singlemovie)
- new load button : type other dataset's directory and type, and click the button



Summary of Final Deliverables

- 1 Status report detailing activity on each of the 4 SOW items. COMPLETED
- 2 Finalized Tiled Video Browser (software) COMPLETED
- 3 Immersadesk4 modifications for SECCHI image viewing (software) COMPLETED
- 4 Improvements/Modifications to Tiled Video Browser as a result of manipulation of SECCHI data once it starts coming in. (software) COMPLETED
- 5 Add cursor interaction to Tiled Video Browser (software) COMPLETED
- 6 Set up and configure Geowall software to handle SECCHI data (software) COMPLETED