Red Tweezers 1.4: Manual

1 Introduction

Red Tweezers is the optical trapping control software developed by the Optics group in the School of Physics and Astronomy, University of Glasgow. A version of this software (with integrated particle tracking code) is distributed by Boulder Nonlinear Systems (BNS) along with their Cube optical tweezers system. All Cube related support questions should be directed to BNS as opposed to Glasgow.

This software aims to provide the most commonly used features of holographic optical tweezers in an easy-to-use package, which also acts as a platform for more sophisticated control algorithms. It includes code to render holograms using OpenGL Shader Language, and can overlay the traps on a video image to provide an intuitive way of manipulating particles with the mouse. It can also be connected to our iPad application which facilitates control of multiple particles in 3D much more simply.

2 System Requirements

This manual assumes that you have:

- A recent graphics card capable of hardware-accelerated 3D graphics (most currently available nVidia and many ATI/AMD cards are suitable, the requirement is that they can execute fragment shader code in hardware). This is necessary such that the GPU can execute the computations required to generate a hologram.
- A spatial light modulator (SLM) and a suitable secondary monitor output (usually DVI-D or dual-link DVI-D) to interface with it.
- A camera that is compatible with the National Instruments IMAQdx driver. This includes most IIDC-compliant FireWire cameras, GigE Vision compatible cameras, DirectShow-compatible USB cameras and many others.
- A multi-core PC (e.g. with an Intel i7 or i5 processor). This is recommended rather than required.
- National Instruments LabVIEW 2010 with the Vision extension. It is possible to run Red Tweezers using the run-time version of LabVIEW, but If you intend to customize Red Tweezers (and most installations other than our own will require at least a little tweaking), you will need a development license for LabVIEW and the Vision Development Module.

2.1 Tested Hardware

This is by no means an exhaustive list, but it may be useful to know the hardware that we have tested most extensively:

- SLM: The software was developed using Boulder Nonlinear Systems 512-pixel SLMs, with DVI control box. It also works with Holoeye or Hamamatsu SLMs, using the alternative shader source VI (included).
- Graphics Card: Our experience is that any recent card will do. However, much of the development work was done on nVidia GeForce 8600gt (now obsolete) or GT330 (more recent). We also use ATI cards, such as the AMD Radeon HD 6490M.
- Camera: We have used the DALSA Genie HM-640 most with this version of the software (it is a fast monochrome CMOS camera with a gigabit Ethernet interface), and in the past have made extensive use of the Prosilica EC1280 FireWire camera (no longer available but close replacements are available from Allied Vision Technologies).

3 Getting Started

The first requirement is to install the necessary portions of the LabVIEW runtime environment, including the Vision runtime library (while the LabVIEW runtime environment is freely redistributable, use of the Vision library requires a license code, obtainable from National Instruments). You should also ensure your

system has up-to-date graphics drivers and a reasonably recent graphics card (the technical requirement is that it can execute GLSL fragment shader programs in hardware, which is the case for all nVidia cards we have tested, and many ATI/AMD cards too).

3.1 Running the programs

Two programs must be run in order to use Red Tweezers to drive the SLM: firstly, hologram_engine_64.exe must be run. This will open a splash screen on the primary monitor which will ultimately display the pattern on the SLM. Next, the LabVIEW interface (red_tweezers_interface.vi, Fig 1) should be opened and run. Upon running red_tweezers_interface (by clicking on the white arrow in the upper left corner of the GUI) the hologram engine window will be relocated to the secondary monitor and a hologram will be displayed within the window. This shows that communication has been established between the two programs.

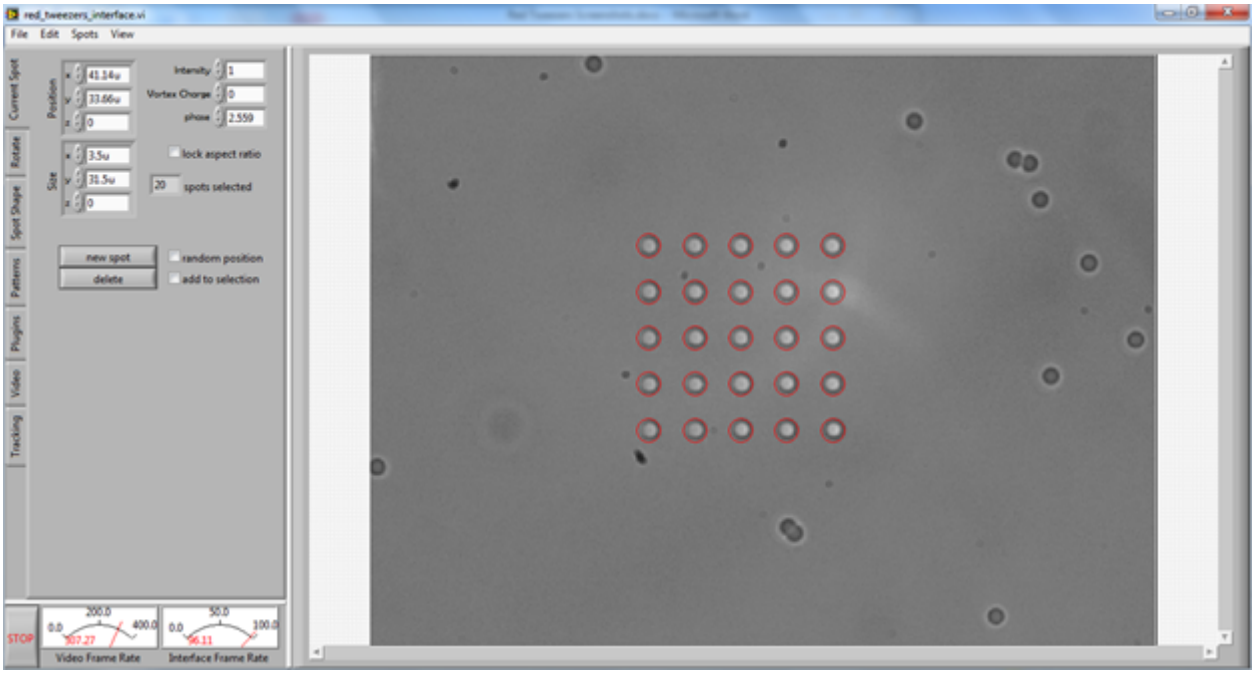

Figure 1 Red Tweezers Interface main GUI

In order to keep the code as simple as possible while maximizing performance, the user interface on the hologram engine (the window that displays the hologram, not the LabVIEW GUI) is extremely minimal. This has led to some confusion over how to close it. **The simplest method of closing the hologram server is to click its button on the Windows taskbar (so that it becomes the active window) and then press <Alt+F4>.** On Windows 7 it is sometimes also possible to right-click the taskbar icon and choose 'close window'.

It is worth noting that some of the Windows Aero effects which are used when switching between windows can cause the hologram display to disappear, meaning that any optical traps currently in use will switch off. It may be worth disabling some of these effects to prevent this (or, indeed, disabling Windows Aero altogether to improve performance).

3.2 SLM Settings

By clicking on the *File/Settings* Menu along the top of the main Red Tweezers GUI, the user can edit the camera settings and the SLM settings. The sections to follow will outline the functionality in each of the tabs of the settings GUI. These settings should be adjusted properly prior to shipping, but if the system setup is modified the user may need to make adjustments.

The GUI that controls the SLM settings is shown in Fig. 2. Each of the tabs available will be discussed in greater detail in the sections to follow.

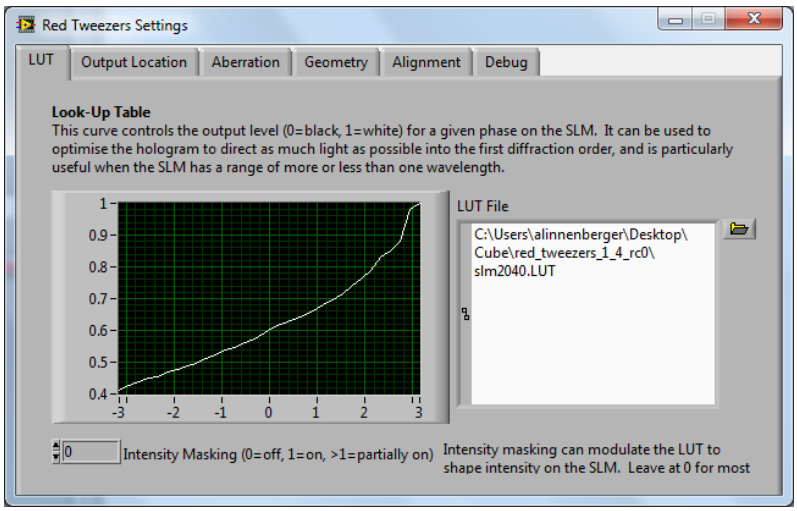

Figure 2 SLM Settings GUI

3.2.1 LUT Tab

Within the SLM linear increases of applied voltage to the liquid crystal layer result in a nonlinear optical phase delay. Thus, it is necessary to generate a calibration of input voltage to output phase delay such that a linear 0 to 2pi phase shift can be realized. This is accomplished through a look up table, which is calibrated at the wavelength that the SLM is designed to be used with. This curve can be viewed in the *LUT* tab of the *SLM Settings* GUI (Fig 2). The location of the default LUT table is shown at right. The user can replace the LUT with their own custom LUT if desired. While the extension of the file is *.LUT, the file is simply a text file that can be opened with Notepad. The file contains two columns of data. The first column in input value which linearly increments from 0 to 255. The second column is the output value which increments non-linearly from some value to 255. The input represents every possible value that can be found in a hologram (0…255 means 0…2pi phase shift), and the output is the value required to realize the desired phase shift.

For more information on generating a custom LUT, contact your SLM manufacturer.

3.2.2 Output Location Tab

In order to display holograms on the SLM, it is necessary to position the hologram engine window on the correct monitor (in the upper left corner of the space immediately to the right of your primary monitor). Within Settings dialog select the *Output Location* tab and adjust the value in the 'Monitor' edit box as shown in Fig 3. If the SLM is connected to the computer then monitor 2 will be available and the user can toggle the red box shown (and thus the location of the hologram) shown in Fig. 3 between the primary monitor and the SLM.

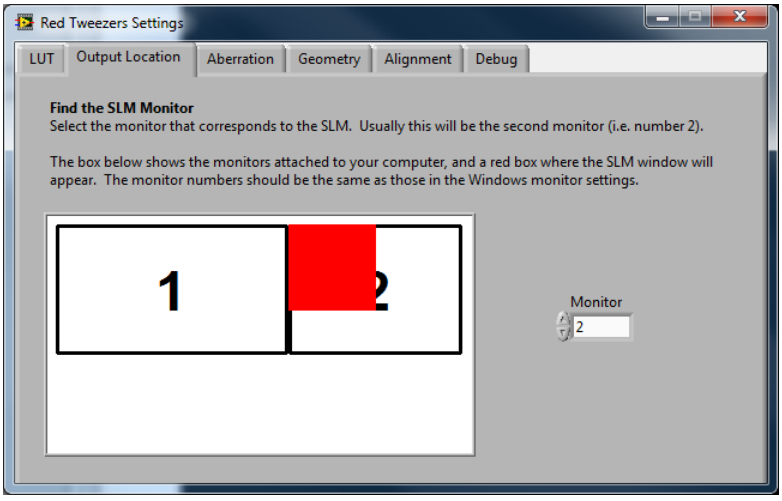

Figure 3 GUI to set the output location of the hologram to properly display the hologram on the SLM

By default, the position of the hologram window is appropriate for a Boulder Nonlinear Systems SLM. To use the program with Hamamatsu or Holoeye SLMs, enable the "full screen" button to display the hologram on the whole external monitor rather than just the top left corner. Also, you will need to swap "shader_source_packet.vi" with "shader_source_packet_nonbns.vi" to output 8-bit grayscale patterns rather than 16-bit red and green patterns.

3.2.3 Aberration Tab

The aberration tab (Fig 4) allows the user to easily correct for aberrations within the SLM, and in the optical trapping system. One can simply observe the fidelity of the back reflection of an optical trap while adjusting the Zernike Coefficients to optimize the beam quality, though it is possible to obtain better results using one of the many aberration correction strategies presented in the literature; Red Tweezers has built-in support for generating Shack-Hartmann type patterns as described in Bowman *et al* (J. Optics **12** 4004, 2010).

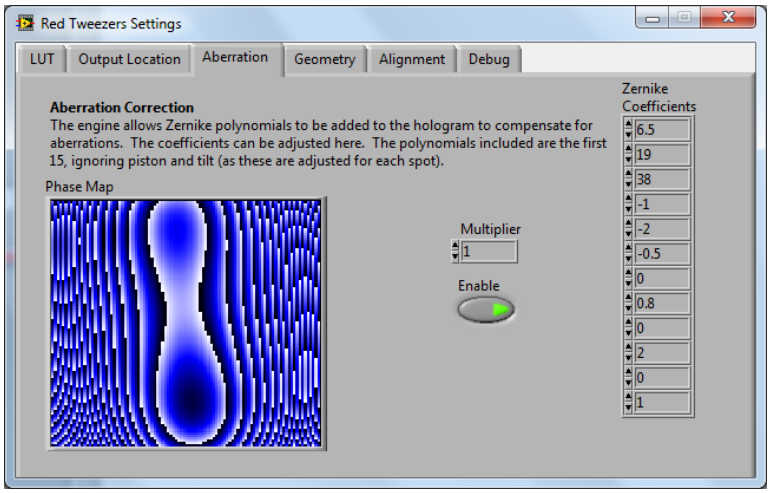

Figure 4 GUI to correct for system aberrations

3.2.4 Geometry Tab

Settings within this tab allow the user to match the software to the physical geometry of the trapping system.

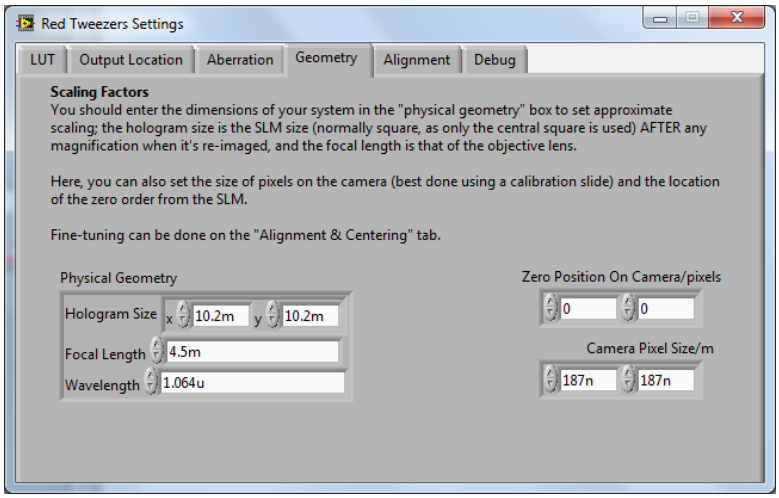

Figure 5 Setting Red Tweezers Geometry

Physical Geometry

To allow meaningful quantities to be used in hologram calculation, the hologram engine asks for some parameters on the scaling factors page. These are outlined below:

- Hologram Size: this is the effective size of the displayed hologram *after* any re-imaging optics. Typically, the SLM is 4f-imaged onto the back aperture of the objective. In this case, the size you

enter here should be the size of the (square) hologram displayed on the SLM scaled by the magnification factor. If the hologram is *h* high, you enter $h*f_{tube}/f_{fourier}$. For example, the BNS SLM is 7.68mm high, and is relayed by a 75 mm lens then a 100 mm lens would give a hologram size of 7.68*100/75=10.24mm.
NB these fields use SI prefixes and are in metres, so 5.4m means 5.4 *0.001 m, i.e. 5.4mm.

- Focal Length: this is the effective focal length of the objective. This is calculated by dividing the focal length of the design tube length (180mm for Olympus, 200mm for Nikon, 163mm for Zeiss usually) by the magnification of the objective (40x for Olympus). Thus, 180/40 = 4.5.
- Wavelength: the wavelength of laser light used.

Zero position this is the position, in pixels, of the zero order spot from the SLM on the camera. This is typically in the upper left corner of the camera, which keeps the 0^{th} order out of the field of view. This can be set most simply by creating a spot at (0,0,0)- e.g. by clicking "new spot" on the current spot tab- and then adjusting this parameter until its marker is on top of the zero order spot.

The camera pixel size is taken by dividing the width and height of the field of view by the width and height of the camera.

3.3 Alignment Tab

In the Alignment Tab the user is primarily concerned with the edit boxes that allow the user to adjust the scaling of the spot positions, and the rotation. Assuming all other parameters are set properly the scaling of the spot positions should be approximately 1 for x, y, and z. However, slight adjustments may be required. To determine what value should be used, place a trap in each of the 4 corners. If necessary, slightly adjust the tip/tilt of the imaging arm of the cube to center the back reflection of the trap in the upper left corner with the red circle. Then tweak the x and y scaling factors such that the traps in the top right, bottom right, and bottom left land in the red circles.

Rotation should be adjusted if the camera is rotated with respect to the coordinates of the software.

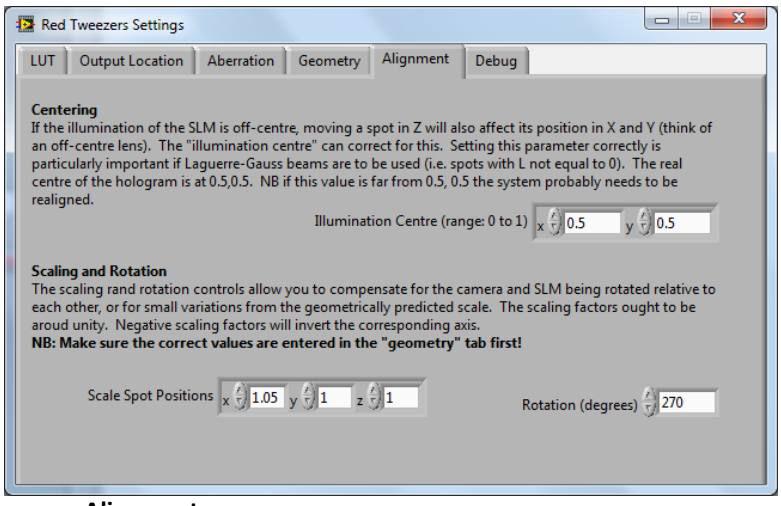

Figure 6 Setting Red Tweezers Alignment

4 Red Tweezers Main GUI Features

Along the left side of the main GUI are a series of tabs including: Current Spot, Rotate, Spot Shape, Patterns, Plugin, and Video. The purpose of each of these tabs will be discussed in the sections to follow. Along the right side of the screen the camera output is overlaid with your trap locations. This allows live, interactive control with the sample.

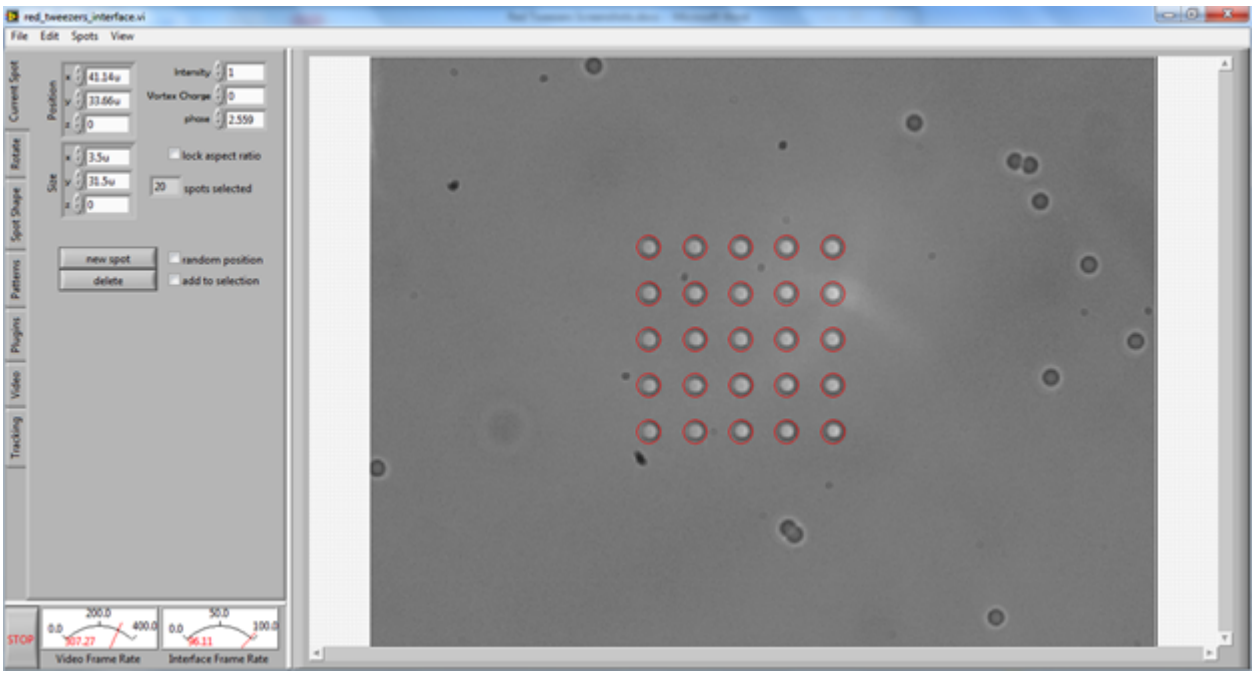

Figure 7 Red Tweezers Main GUI

4.1 Current Spot Tab

The "Current Spot" tab is the default tab, and displays the 3D position of the current spot. This encapsulates most of the functionality of many older versions of our software- though it now operates on all selected spots rather than just a single spot.

At any point in time spots can be added or removed from the "current spot" tab in the toolbox. It is also possible to add spots by double-clicking the image, and to delete them from the edit menu or by pressing <Ctrl+Delete>. To reposition spots, first select them (by clicking) then either drag them on the image or adjust the "position" controls in the current spot tab. Multiple spots can be moved or deleted at once- just click on them while holding down shift.

The simplest way of selecting a trap is to click on it. However, it is also possible to select traps from the Edit menu- selecting all spots is possible via Select All Spots (Ctrl+A), and spots can also be selected by index or by group. When selecting by index or group, the traps are overlaid on the video image by their respective ID (be it index or group number).

Grouping is accomplished from the Spots menu- the group mechanism in Red Tweezers is very simple, as each spot belongs to at most one group. Choosing "Group selected spots" will create a new group containing all the currently selected spots, which are removed from their existing groups (if any). Similarly, "ungroup" will remove all the spots from the group. It is not currently possible to create groups of groups- i.e. there is no group hierarchy.

If multiple spots are selected, holding down shift while dragging scales and resizes the spots around the center of the pattern.

When multiple spots are selected, the "size" field will display a nonzero value. Adjusting this allows the spots to be scaled about their center of mass, either one dimension at a time or uniformly in three dimensions (depending on whether the "preserve aspect ratio" checkbox is selected). There is also an indicator to show how many spots are currently selected.

Holding down "shift" and dragging will map the mouse movements into 2D scaling and rotation about the center of mass. This also rotates and scales line-trap vectors.

The selected spots can be translated by editing the X, Y and Z positions. The position is the center-of-mass for all the selected spots (i.e. the mean position) if more than one spot is selected. This performs the same action as dragging the mouse on the image.

The intensity of the currently selected spot(s) is adjusted with the "intensity" control. This scales the intensities when multiple spots are selected, and displays the mean intensity- i.e. if spots with intensities 0.5, 0.3 and 0.7 are selected, it will display 0.5, the mean. Changing this to 1.0 would double the intensities, to 1.0, 0.6 and 1.4, such that the new mean is 1.0.

Each spot has a "phase" property as well- when generating the hologram, we effectively superpose a number of plane waves, one for each spot. The phase property of each spot (defined here in radians) sets the relative phase of these plane waves and can in some cases be optimized to reduce ghost orders or create more uniform arrays of traps. When multiple spots are selected, this displays the mean phase, and changing the phase control adds the same constant to all phase terms. The phase can have values with the range of $-\pi$ to π, though its effect is modulo 2π.

4.2 Rotate Tab

Spots can be rotated from the "rotate" tab, which allows the controlled rotation about given axes by a specified number of degrees. When this tab is visible, a rotation marker appears at the center of mass of the selected spots. Dragging outside of this marker performs 2D rotation around it. Dragging inside rotates in 3D, using the metaphor that the circle on the screen is really a sphere, and we are dragging its surface to rotate in 3D.

4.3 Spot Shape Tab

Red Tweezers includes the possibility of creating spots which are not simply diffraction-limited foci. On the Spot Shape tab, it is possible to create a circular aperture (applying to that trap only) on the SLM, with control of its size and position. By using multiple apertures with weighted intensities, it is possible to create more complicated shapes such as annular apertures (R. W. Bowman, G. Gibson, and M. Padgett. Particle tracking stereomicroscopy in optical tweezers: Control of trap shape. Opt. Express, 18(11):11785–11790, 2010) The units used here are SLM widths, centered on the center of the SLM (technically the center entered in SLM Settings, rather than the actual center, so we respect any shift that has been added for alignment purposes). Thus, the SLM goes from -0.5 to 0.5, and a radius of 0.5 corresponds to a circle which fits exactly onto the square hologram. By default, spots have radius 1- which means there is no visible aperture.

Code for generating line traps is also present- using a generalization of the algorithm presented by Roichman (Y. Roichman and D. G. Grier. Projecting extended optical traps with shape-phase holography. Opt. Lett., 31(11):1675–1677, June 2006.). Our line traps extend along an arbitrary 3D vector, and are created by amplitude-masking the hologram with a sinc function. There is a "phase gradient" parameter, which effectively translates the sinc function on the SLM. Its range is -1 to 1, where +/-1 corresponds to the Poynting vector aligning with the line trap and 0 to the Poynting vector being normal to the line trap. This is important both as it controls the scattering force along the line, and as it is crucial to efficiency (for example, a line trap with a significant incline in Z may not be possible with zero phase gradient, as this might take the bulk of the required light outside of the available numerical aperture).

4.4 Patterns Tab

On the "patterns" tab, it is possible to create patterns such as Shack-Hartmann arrays (R. W. Bowman, A. J. Wright, and M. J. Padgett. An SLM-based Shack–Hartmann wavefront sensor for aberration correction in optical tweezers. J. Optics, 12(12):124004, 2010.), which are useful for aligning the system, and measuring wavefront aberrations.

4.5 Plugins Tab

ASI Stages

This plug in is used to control motorized XY or XYZ microscope stages from Applied Scientific Instruments inc.

iTweezers
Red Tweezers is supplied with a plugin that is the server for iTweezers. This listens on two TCP ports for a connection from our iPad app, iTweezers, and then handles streaming video to the device and two-way sync of trap coordinates from the device. The iTweezers plugin makes use of some of the advanced plugin features, such as initialization and shut-down code run when Red Tweezers is started and stopped. This allows iTweezers to listen in the background, when the plugins tab is not visible.

MightexLED
This plug in is used to control the brightness of the illumination system for the Cube. If the iris on the condenser is open, and the condenser is properly positioned above the sample, then it is generally not necessary to increase the brightness of the illumination above 200.

Writing new plugins
Creating new plugins can be extremely simple- a folder or LLB called pluginname (or pluginname.llb) will cause an entry "pluginname" to be displayed in the plugins list. Said folder should contain at least a file called "pluginname_interface.vi", which is the front panel for the plugin, displayed in the subpanel on the plugins tab. When the plugin is visible, this VI is run once every time the main loop of Red Tweezers executes. For more advanced plugins, it is possible to put two additional VIs in the folder, pluginname_init.vi and pluginname_close.vi. These are run when Red Tweezers starts and stops respectively, irrespective of whether the plugin is visible at the time. These could be used to set up communications to a piece of equipment in the background, or to run another VI in the background as a server. If the interface VI is currently running or not runnable for another reason, no error message is displayed- hence iTweezers can run the interface VI in the background without issues. Ancilliary VIs and other files in the plugin folder are simply ignored, and can be used without problems.

5 Loading and saving settings

To avoid the problem of having to constantly "make current values default" and save, Red Tweezers employs an automated settings-saving mechanism. This iterates through all the VIs in memory, and for all those which have the string "[Red Tweezers Save Settings]" in their description it extracts the label and value of each control. This information is saved in an XML file when requested from the "File" menu. The location of the last used XML file is noted in the Windows Registry, and this file automatically re-loaded when Red Tweezers is started for the first time, or whenever it is recompiled. The user is prompted to save settings to this file whenever the "stop" button is clicked.

6 Setting up a camera

In order to keep the code as simple and easy to modify as possible, we have opted to provide several interchangeable blocks of code to communicate with cameras. These allow the use of cameras and framegrabbers compatible with National Instruments IMAQ and IMAQdx drivers, which include most FireWire and gigabit ethernet cameras which comply with the IIDC or GigE Vision standards. By default the software uses a stub camera driver that generates a blank image, but by changing the active page of a diagram disable structure at the bottom of the main Red Tweezers block diagram, it is trivial to switch to a different driver. This can only be done when the program is not running, however we've prioritised easy-to-read code over UI polish in this case. Comments in the block diagram explain the various options available.

Other than choosing the camera driver to use, it is necessary to set up some constants (again on the block diagram) such as the ID of the camera to use and optionally the region of interest. These differ for each block of camera code, and are documented in the block diagram.

7 Customizing Red Tweezers interface

7.1 Coordinate systems

As far as possible, the coordinate system used should be “real coordinates”, in units of meters relative to the defined zero position. This is then scaled by the settings for the SLM, and used for trap positions. The relationship between real coordinates and camera pixels is defined in the camera settings, and the coordinate conversion VIs (coords_real_to_image and coords_image_to_real) are the preferred way of converting between them. Using global variables directly is discouraged as the format may change and break your code in future versions.

As the user may change the displayed image without warning, we recommend you always store positions in “real coordinates” and work in them whenever possible. Image coordinates should really only be used when drawing to, or reading from, the camera image.

7.2 Important quantities

7.2.1 Spot Parameter

This is perhaps the most important quantity of all. This is passed around the Red Tweezers main loop every time and is sent to the hologram engine to create the spots. It is an array, with exactly one element for each spot, which describes the various parameters relating to that spot. A global variable, "spots" allows other programs to modify the spots in Red Tweezers, if the main loop has not modified the spots variable then the contents of the global variable will override the current state of the main loop. A *typedef* spot.ctl defines the elements, which are currently:

- **Position:** the location of the trap in real coordinates, in 3D
- **Intensity:** the relative power in the trap. Note that this is not guaranteed to be linear, the algorithm that implements it is very simple!
- **Vortex Charge (l):** a nonzero value of l adds a spiral phase to the spot, giving it orbital angular momentum. This is constrained to have integer values.
- **Phase Offset:** a constant phase shift applied to the spot. This corresponds to translating the grating on the SLM, and is sometimes useful for reducing the power in unwanted traps.
- **NA Position:** This is a three-element cluster, defining a circular aperture, which is applied to the spot in the SLM plane. This allows the NA of a particular spot to be artificially reduced, and for the direction of light in a low-NA spot to be controlled (by shifting the aperture on the SLM we change the direction from which light is focused into the trap). The elements are:
    - **x** position on the SLM, in units of the SLM width, from 0 to 1
    - **y** position in the same units
    - **Radius** of the aperture, in the same units, default 1. Note that 0.5 produces a circle which fits exactly into the central square of the SLM.
- **Line Trapping:** By modulating the grating to produce a particular spot with a since function, it is possible to produce a line shaped trap. This cluster of four elements is described in more detail under "spot shape", but in brief it contains
    - **x,y,z** vector along which the spot should be smeared out to create a line of light
    - **Phase gradient** a number between 0 and 1 which sets the phase gradient along the line, i.e. the dot product of the mean Poynting vector of the light and the line trapping vector, normalised to lie between 0 and 1.

7.2.2 Selection and Groups

This should be an array of the same length as spots. Each element is a cluster (again a typedef, selection_element.ctl) with two elements,

- **selected** is a Boolean telling us whether that spot is currently part of the selection. This affects various operations in Red Tweezers as well as causing the spot to change colour on the video image.
- **Group** is an integer, which is set to the ID of the group to which the spot belongs, or zero for ungrouped spots. New groups are allocated the first nonzero integer which is not already in use by a

group, and each spot belongs to at most one group, i.e. there is no support for nested groups. Typically groups are used only to change which spots are selected, and the operations are performed on the selected spots. However it is in principle possible to perform operations on specific groups instead.

7.2.3 Camera scaling parameters

These parameters reside in red_tweezers_globals_internal.vi, and are currently the position of the zero order spot on the camera, in pixels, and the size of pixels (in x and y) in meters, taking account of the magnification of the system. They should not be used to scale from one coordinate system to another, for that you should use the coordinate conversation VIs. If you implement a custom camera VI, however, you might want to set these quantities from that VI, as they will no longer be shown when you click "camera settings".
NB the way scaling between the camera and real coordinate systems is defined may change in the future, which is why the coordinate conversion VIs are the **only** recommended way of converting.

7.2.4 Camera ROI

These parameters reside in the same global variable as above, and should tell us the current region of interest being acquired by the camera. It is a standard 4 element rectangle as used by the IMAQ functions, i.e, [left, top, right, bottom]. If the first two elements are nonzero, they will cause markers to shift on the image. This is done so that, if the camera acquisition region is shrunk down to enable high speed imaging, the spot markers still appear in the correct place.

7.2.5 Mouse information

This information is passed into the case statement corresponding to each tab, allowing custom handling of mouse actions. It contains a number of hopefully self-explanatory elements, including the current and previous positions of the mouse cursor on the image, whether the mouse button is down, and which modifier keys, if any, are pressed. Note that the mouse position is already converted into image pixels rather than screen pixels, so no account need be taken of the zoom factor of the image.

7.2.6 Image_overlay_extras

This allows custom overlay items to be drawn on the image, from the top loop. See the middle loop (responsible for displaying the image and rendering the overlay) for more details.

7.3 Adding tabs

Red tweezers has been designed with extensibility in mind, and to that end there is a relatively well defined procedure to add a tab, with associated functionality, to the interface. Note that most tab functions can be performed effectively using plugins, and in the future strict plugin typedefs will make plugins and tabs nearly equivalent.

Firstly a new page should be added to the tab control on the front panel, and named appropriately. The case statement in the "timeout" case in the main loop should be extended with a case for the new tab, most easily done by using the "create case for every value" option. The minimum requirements for a case in this case structure are that spots should be passed in and out again. If you do not want to modify the spots for selection, simply wire the two quantities through the case statement. It is also important to have some kind of wait statement in the case, usually this is a 10ms delay. Not including a delay will lead to the top loop consuming all the available processor power and slowing down other processes, such as the camera loop.

The desired controls and corresponding block diagram functionality should then be added to the case. Our hope is that it will not be necessary to wire many other things into and out of the cases, the terminals going into and out of the case structure should be sufficient to accomplish most tasks.

7.4 Adding menu items

Menu items are added in a very similar way to tabs, in that once the item has been added to the menu (via edit->run time menu), it is simply a matter of adding a case to the case statement in the "menu item" event case in the main loop. Again, the minimum requirement is that the spots and selection are wired through,

but there is no timing requirement on menu items as they will not cause the loop to run out of control if they terminate immediately. It is also not possible to respond to mouse events from within a menu case, as this does not really make sense!

7.5 Useful subVIs

Coordinate conversion: As mentioned above, there are subVIs available (prefixed coords_) to convert between coordinate systems, and these should be used in preference to reading the global variables that set up coordinate conversion, to guard against incompatibility with future versions of Red Tweezers.

UDP Sender Generic: this VI takes the spot parameters, performs coordinate conversion and some scaling, and then communicates with the hologram engine. Described in more detail below, this is the VI to use if you want to control our hologram engine without using Red Tweezers in its entirety. Remote control is also possible by setting the "spots" global variable, which is more flexible and easier to debug, at the expense of being slightly slower.

Extract_position & replace_position simplify working with just the position information from spots. This often removes the need for bulky unbundle/bundle VIs.

Find_centroid calculates the mean position of the selected spots. This is useful for a number of purposes, and is used in various places in Red Tweezers, from finding the mean position for the "current spot" tab to calculating the rotation center when transforming spots.

8 Hologram engine

Central to the ability to control holographic optical tweezers is the generation of appropriate holograms. In the case of Red Tweezers, this is done on the graphics card, using a stand-alone executable file written in C, which communicates with LabVIEW via UDP packets.

As it is possible to recompile the code which is run on the graphics card without recompiling the C source, this hologram engine serves as a general purpose toolkit for calculating functions and displaying them on the screen. It is our hope that this will find use in many SLM control applications and perhaps other uses as yet undiscovered.

The UDP packets used to control the hologram engine consist of human-readable, XML-style text, which should be simple to generate from any programming language. Examples of this are not supplied, but it is our hope that the documentation in this section will be sufficient to implement the functionality of "UDP Sender.vi" in the programming language of your choice.

8.1 Overview

As rendering holograms is a computationally intensive problem, involving significant amounts of floating point calculation and then transferring large arrays to the graphics card for display, it makes sense to generate the holograms on the graphics card directly. Rather than using general purpose toolkits such as CUDA and OpenGL, we have opted to render the holograms in OpenGL, as we are effectively dealing with a graphics problem- rendering a function to the screen as quickly as possible. This has the advantage of compiling into a self-contained executable with no dependencies on CUDA runtime libraries, and of being able to recompile the CPU code on-the-fly. Its disadvantage is the requirement to render holograms in a single pass, i.e. the value of each pixel can depend only on the "uniform" parameters, and it's position in the hologram.

8.2 UDP communication

In order to maximize portability and speed, as well as to free us from a number of window-management quirks, the GPU code runs as as a separate process, accepting input via UDP. This latest version accepts the

shader language source code at runtime, also via UDP, which allows the user to tweak or replace completely the code which is executed on the graphics card.

Normally the hologram engine is run on the same computer as the rest of Red Tweezers, in which case the loopback network interface is used, i.e. UDP packets are sent to localhost, 127.0.0.1, port. It is possible to run the hologram engine on another computer by simply entering another IP address. Doing so will incur a slight performance overhead, but it should not be too significant if the network connection is fast.

Support for IPv6 is nominally included, but is untested. It might therefore require some tweaking if an IPv6 network is to be used.

8.3 Commands

Most of the work in the hologram engine is triggered by UDP packets that look like this:

```
<data>
<uniform id=2>
1.9 37.3 4657 0 465
</uniform>
<uniform id=3>
42
</uniform>
<uniform id=7>
0 0 0 1
</uniform>
</data>
```

This demonstrates the structure of a UDP packet (quasi-XML commands enclosed in the data tags) and also shows the most commonly used command, which sets the value of a "uniform" parameter to the shader program. Each time the hologram engine receives a packet containing uniform values, it re-renders the hologram. Normally, each time the UDP sender is executed, it sends just such a packet, to change the spot parameters and update the SLM.

The next most important command is <shader_source>, which sends the shader program to be compiled. On receiving this command, the shader program is recompiled with the new source. If the source contains errors, the hologram will not update, but no error message is shown. This may be implemented in future versions, however for now it is a consequence of the deliberately very minimal UI in the hologram server, to maximize performance. The uniform parameters are declared at the start of a shader program, and are parsed by the hologram engine before the source is compiled to match up the <uniform> commands to their variables. Thus, it is not necessary to separately define the uniform parameters for a shader program, merely to note the order they are defined, as parameters are updated by specifying the ID of the parameter, which is its position in the list of declarations.

The remainder of the commands define parameters which affect the behavior of the output window and other options, listed in full below. Note that italicized items in syntax examples are parameters.

Command	Syntax	Description
Set the value of a uniform parameter **and** render the pattern.	<uniform id="*n*"> *num1 num2 num3 ...* </uniform>	Sets the value of the *n*th uniform variable (in order of declaration in the source, starting from zero). Depending on the data type, this may have between 1 and 200 numbers, separated by spaces. Decimal points are acceptable, and any numbers not required by the data type (e.g. a float will require only one number, a vec3 3 numbers and an array could be many more). The presence of this tag also re-renders the pattern.
Set the value of a texture parameter **and** render	<texture id="*n*" width="*w*" height="*h*" format="*format*"> *num1 num2 num3 ...* **or** <binary size="*s*">*...* </binary> </texture>	Sets the value of the *n*th uniform variable, which is a texture. This must contain *4*w*h* numbers, four per element in the texture (all textures are RGBA format). It is possible to include a blob of binary data instead of a human-readable list. In this case, the binary data must be enclosed in tags specifying its size as shown, and the optional "format" parameter set to "packedu8" or "packedfloat". See the "texture_test_packedu8.vi" example.
Set the shader source and recompile	<shader_source> *#openGL shader language source code* </shader_source>	The text between the two tags (which may contain newlines, but should not contain the closing XML tag) is compiled to form a new fragment shader program, and used for future rendering. N.B. If the shader source contains syntax errors, the pattern will simply stop updating. Helpful error messages are planned for future releases.
Change the aspect ratio of the pattern	<aspect> *AspectRatio* </aspect>	The pattern is rendered at a fixed aspect ratio (1:1 by default) regardless of the window size, resulting in black bars if it is not square. This command changes the shape of the region into which the pattern is drawn (height/width). N.B. This command may not take effect until the window is resized.
Move/resize output window	<window_rect> *X, Y, W, H* </window_rect> **or** <window_rect> all monitor *M* </window_rect>	Make the output region either the rectangle with top left corner *X, Y* and size *W* by *H* pixels (all numbers are integers), or make it full screen on monitor number *M.* N.B. this does not change the aspect ratio of the pattern, if you want to do that then you also need to use the "aspect" command. It is ok to use the aspect and window_rect commands in the same UDP packet.
Lock/unlock redraws to vertical sync	<swap_buffers_at_refresh_rate> *1* ***or*** *0* </swap_buffers_at_refresh_rate>	This sets whether to synchronize updating the hologram with the monitor's refresh rate. Doing so (1) is usually best, and eliminates the possibility of "tearing" by buffering a couple of frames in advance (this is default on most systems). Not locking (0) decreases latency but can allow tearing.
Enable/disable two-way communication	<network_reply> *1* ***or*** *0* </network_reply >	If set to 1, each UDP packet will be answered by the hologram server, with some timing information. This can be useful if you need to synchronize with the holograms being displayed. N.B. if the hologram updates are synchronized with monitor refresh, the reply is sent when the buffer is swapped, which usually happens a couple of frames before it is actually displayed. Also, if you send more UDP packets than there is time to deal with (usually only an issue with very complicated patterns or very fast update rates) some of them get ignored, and those ones will not get replies.

8.4 Timing

One of the great advantages of GPU computing is that it can be extremely fast. However, displaying patterns on a monitor has constraints other than speed: because the monitor updates at regular intervals, we must take care to update the pattern in between refresh cycles, so as to avoid "tearing". This is the name given to the effect seen when the pattern on the screen is changed halfway through a monitor refresh cycle. The result is that the top part of the monitor shows one frame, while the bottom part shows the next frame. To prevent this effect, it is usual to render into an off-screen buffer and then switch from the currently displayed buffer to the just-rendered one for the next monitor refresh cycle. This "double buffering" is the norm on most systems, and improves the appearance and performance of 3D games, etc. However, the price for this improvement is latency; it takes a couple of frames (i.e. about 30ms on most monitors) for the pattern to be displayed on the monitor, once it is rendered. In applications where latency is a problem, it might therefore be desirable to decrease latency even at the expense of tearing. The hologram engine can request that the graphics card drivers enable or disable double-buffering with the `<swap_buffers_at_refresh_rate>` command, however this may be overridden by graphics card drivers. If it is important to control whether double-buffering happens in your application, we recommend you check how this parameter is set: usually the graphics card control utility will allow you to force it on or off, or allow programs to control the setting. The command will have an effect only if the last option is chosen. In Red Tweezers, double buffering is turned off by default. To enable it, you will need to edit the "shader_source_packet.vi" script, to change the setting in the initial packet sent to the SLM.

8.5 Shader program design

While it is outside the scope of this manual to give a course on OpenGL Shader Language for scientific computing, I will aim to describe how the hologram engine goes about rendering functions using the shader program.

Shader programs are typically used by high performance computer games to provide more realistic lighting effects in 3D scenes. They are implemented as small programs, often termed "computational kernels", which are executed once for each pixel on the screen. No communication is allowed between the instances of the program corresponding to different pixels; the only input allowed is the uniform variables (which are the same for each pixel) and some parameters relating to the scene as rendered so far (the only quantity which concerns us here is the position on the screen, gl_TexCoord[0].xy), and the only output is the colour of the pixel on the screen.

A typical hologram-rendering shader program might look like:

```
uniform float k;
uniform float f;
uniform vec2 slmsize;
uniform vec4 spots[10];
uniform int n;
void main(){
// basic gratings and lenses for a single spot
  vec2 uv = gl_TexCoord[0].xy*slmsize;
  vec3 pos = vec3(k*uv/f, k*dot(uv,uv)/(2.0*f*f));

  float phase, real=0.0, imag=0.0;
  for(int i; i<n; i++){
    phase = dot(pos, spots[i].xyz);
    real += spots[i][3] * sin(phase);
    imag += spots[i][3] * cos(phase);
  }
  phase = atan(real, imag);
  float g = phase / 6.28 + 0.5; // map -pi to pi onto zero to one

  gl_FragColor=vec4(g,g,g,1.0);
}
```

The first few lines of this program define the uniform variables, our input. Here, we have five parameters. The first two (with ID 0 and 1) are simple floating-point numbers. It is worth noting that the GPU will often only use single precision floating point, so very large or very small numbers may produce unpredictable results. Uniform number 2 is a vector with two elements, so two numbers should be supplied when updating this one. Vector types are always floating point. Uniform 3 is a little more complicated, it is an array of vectors. Arrays of any type can be declared by appending their length in square brackets. When setting array parameters, the length of the (one dimensional) list of numbers supplied in the <uniform> tag is either the length of the array (for float and int arrays) or the length of the array multiplied by the length of the vector (for example, the spots array in the example above will have 40 elements).

This program first extracts the coordinates on the pattern (gl_TexCoord[0].xy) which run between 0 and 1, and scales them to represent physical distance. This is the final input parameter to the hologram calculation algorithm, which is a weighted superposition of plane waves, each with optional lens terms to allow 3D control of the focused spots in the Fourier pane of the SLM. The output of the calculation is then placed into the vec4 quantity gl_FragColor (which has elements red, green, blue, alpha) for output to the screen.

When optimizing shader programs, it is important to remember that they are different in many ways from programs to be executed on the main CPU. Memory access in particular is costly, so it pays to avoid using too many large arrays as uniform parameters. Looping and branching structures are also very costly, and should be avoided wherever possible. It is worth pointing out that using a variable to set the number of iterations of a for loop is not universally supported, some older graphics cards might fail to compile the example above. In that case, for this example one could fix the number of loop iterations at 10, and simply set the weighting term for unused spots to zero.

The shader program used by Red Tweezers is somewhat more complicated, but follows the same general form. It is visible as a large string constant in UDP Sender.vi. A number of example scripts are provided in the “extras” folder for anyone interested in using the hologram engine for other tasks.

The hologram engine is supplied compiled for 32 and 64 bit Windows, and has been tested on 32 bit Windows XP and 32 and 64 bit Windows 7. Note that updating your graphics drivers to the latest ones for your operating system is recommended for best results. The source code for the OpenGL hologram engine is also available if you would like to modify it beyond what is possible with UDP commands.