Interactive Visualization for Light Transport Matrices

Jiamin Bai*
The University of California,
Berkeley

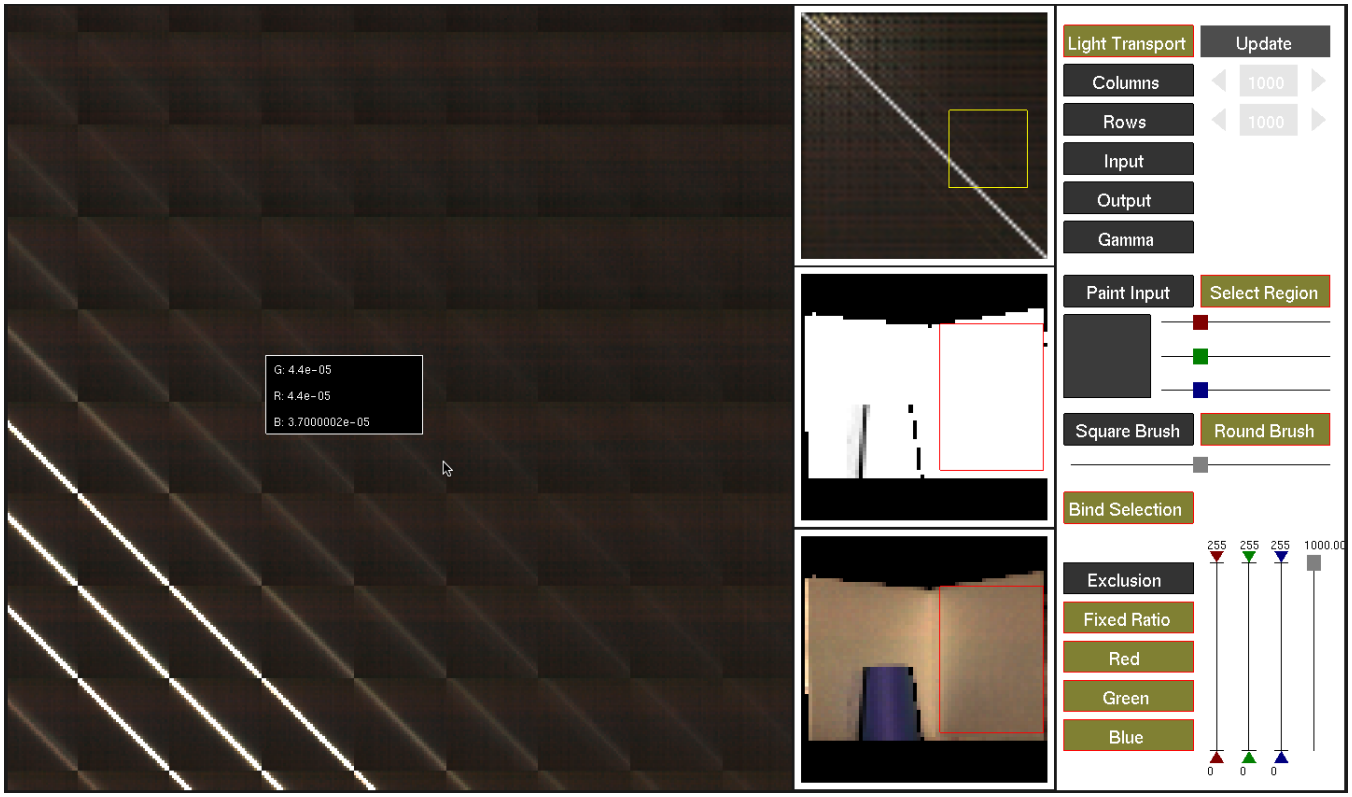


Figure 1: *A screen capture of proposed visualization tool for Light Transport Matrices. The main window shows a sub light transport matrix generated from a set of selected light inputs and observed pixels. The value of the element the mouse is pointing to is displayed at an offset from the mouse pointer. The largest window is the main window and it displays what the user is interested in exploring. The top small multiple provides an overview for the light transport matrix, the middle multiple provides an overview for the input lights and the bottom multiple provides an overview on the observed scene lit with the input lights. These small multiples also act as selection tools for the user. The panel on the far right allows the user to select modes and display parameters.*

Abstract

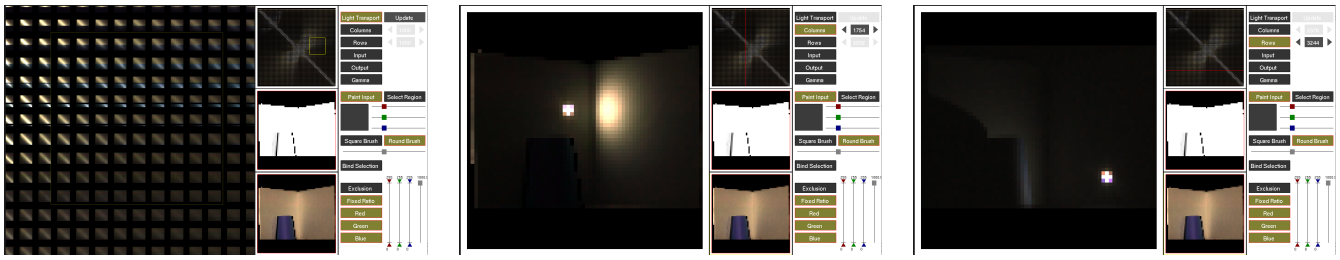
Acquired Light transport matrices are typically large (50 million entries in 3 channels) and prone to noise. Current tools and techniques to inspect these matrices are rudimental and do not elucidate structures and patterns in these matrices. In this work, we design and implement an interactive visual tool that allows the user to interactively explore the data. Multiple windows help the user select the range of lights and observed pixels to generate sub transport matrices in interactive rates. Zooming, panning and value remapping is also supported in our implementation in real time. The key challenge in this work to provide a intuitive tool that enables the user to explore the large database interactively, reducing both the gulf of execution and gulf of evaluation. We use OpenGL and graphics hardware acceleration to render the data in real time.

Keywords: Visualization, Interative, Light Transport Matrices

*e-mail: bjiamin@eecs.berkeley.edu

1 Introduction

Light transport matrices have recently been introduced to the graphics community as the foundation for image-based rendering [Ng et al. 2003] and radiometric compensation [Ng et al. 2009]. While the fidelity of the transport matrix is crucial for accurate rendering and compensation, there has not been any tools developed that facilitates the verification of acquired transport matrices. This is certainly the case for radiometric compensation where one seeks to invert the light transport matrix because the inversion process is very sensitive to small deviations or noise. Visualizing the light transport matrix is commonly done in Matlab, where one would inspect the constructed matrix as an image. As such, selecting regions and remapping color values are perform through several command line instructions. This results in a large gulf of execution and evaluation [Norman and Draper 1986]. Mapping the intent of the user to the commands are not easy even for experienced user. One would have to first find the exact coordinates and values for remapping. It is not uncommon for the user to perform several iterations or refinements before getting Matlab to display what he wants. Errors



(a) Light Transport View (b) Column View (c) Row View

Figure 2: *Here we show 3 possible views the main window supports for exploring the light transport matrix. Notice how layered information on the small multiple 'Light Transport' provide reminders and context to what the user is exploring.*

in the matrix are often caused by corrupted input images which are difficult to locate using Matlab. Other software such as IrfanView allows rapid viewing of the source images but not High Dynamic Range composites in matlab's format. While these HDR input images are the columns of the transport matrix, extracting them individually and inspecting them in Matlab is impractical.

With this in mind, we design and implement a tool that allows the user to interact with the data, performing a large suite of queries using intuitive controls with the mouse. The key to a successful tool would be to provide near instant feedback or updates to the user's queries. This is particularly challenging given the large size of the transport matrix (50 million elements in 3 channels).

2 Related Work

While there has not been work done in the domain of visualizing transport matrices, there are several that deal with visualizing large complex high dimensional data such as LifeLines [Plaisant et al. 1998] and SpotFire [Ahlberg and Shneiderman 1994]. The mantra for designing these highly successful commercial visualization tools is aptly described as: "**Overview first, zoom and filter, then details-on-demand**" [Shneiderman 1996].

To provide an overview of the data, these visualizations have a main window where the data of interest is displayed. Multiple auxiliary windows provide overviews for the data in specific sub domains. Panels accompanying the main window provides tools for searching and filtering the data. For data with numerical features, sliders can be used to efficiently specify the range in which the user is interested in. To this end, the Alphaslider was introduced [Ahlberg and Shneiderman 1994]. Interactivity, which is crucial to reduce the gulf of evaluation, is achieved by having immediate and continuous display of results [Ahlberg and Shneiderman 1994]. Smooth animations are used when exploring the data, either by zooming or panning, provides context to the user.

In other implementations such as GGobi, multiple windows displaying the same data in different domains allows the user to perform domain specific selections (brushing) which are then propagated to the other windows (linking) [Wills 1995]. Brushing and linking also helps to reduce the gulf of execution as the user can intuitively select relevant data points efficiently.

To provide more information for complicated data, layering can be used to superimpose secondary information. It is crucial that this layering is done in a non-intrusive way [Tufte 1990]. If it is intrusive, this additional information will only hinder the ability for the user to efficiently comprehend the original data.

Using these design principles, we propose a design for our Light Transport Visualization as shown in Fig. 1.

3 Design

3.1 Main Window

Emulating the success of SpotFire, we will have a main window where it will display the data of current interest to the user. The main window will be interactive, allowing the user to select regions to focus on. Animations will be used during zooming or panning to provide the user with context and a continual display of results. During the selection process, the span of the selection is layered over the data by drawing a bounding box of the selection. When the selection is confirmed, the bounding box is animated along with the data and fades away, allowing the user to focus on evaluating the data thereafter. Secondary information for the data points (color values) is layered over the data when activated by toggling a key on the keyboard. Color values for pixels directly under the mouse is layered over the data at an small offset from the mouse. This allows fast visual queries as the user will not have to look between two locations on the screen.

There will be 5 different modes for the main window, each to display the data in different domains. We show 3 possible views in Fig. 2.

1. **Light Transport Matrix**
   The current light transport matrix constructed using the set of selected input lights and observed pixels is displayed.

2. **Columns**
   The selected column of the current light transport matrix is displayed in the pixel domain. i.e. observed image under the selected light only.

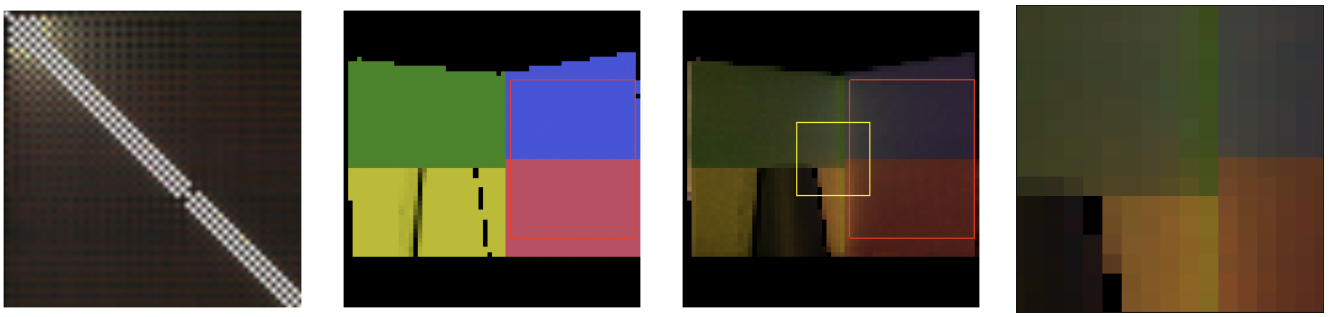
3. **Rows**
   The selected row of the current light transport matrix is displayed in the input light domain. i.e. visualization of the contribution of each light for selected observed pixel.

4. **Input**
   This is just the set of lights that generated the light transport matrix.

5. **Output**
   This is the observed scene due to illumination from all the lights.

3.2 Small Multiples

The light transport matrix characterizes the relationship between a set of lights shining onto a scene (columns of the matrix) and a set of pixels that measure the light leaving the scene (rows of the matrix). As such, different sets of lights and pixels will produce



(a) Light Transport (b) Light Input (c) Observed Scene (d) Main Window

Figure 3: *Here we show a close up of the small multiples and the main window when we are exploring the Observed Pixels. The red bounding box shows the selection of input lights and observed pixels that are used to generate the light transport matrix. The yellow bounding box shows the region that is displayed in the main window.*

different light transport matrices. We employ the use of small multiples to enable the user to create smaller light transport matrices for closer inspection dynamically and easily. For our purposes we will have 3 small multiplies namely, 'Light Transport', 'Light input' and 'Observed scene' as illustrated in Fig. 3

3.2.1 Light Transport

In the first small multiple, we display the current light transport matrix that is constructed by the set of input lights and the observed pixels. This multiple serves three functions; to provide context on which region the main window is showing, to select regions to zoom or pan and to select which rows or columns to inspect. In Light Transport mode, mouse actions will be translated to zooming and panning for the main window. This allows the user to quickly identify local regions he wishes to inspect and efficiently execute the queries. A bounding box is layered over the region that the main window is displaying. This gives the user a gentle reminder to which region of the light transport matrix he is inspecting. However in Columns and Row mode, mouse clicks will be interpreted as selection for respective columns or rows. In this mode, the user clicks on the column or row that he is interested in and the main window will update accordingly. This provides very intuitive range selection as there is a direct mapping from what you see and want to the corresponding mouse actions. The corresponding column or row will also be highlighted to visually remind the user what he is exploring. Key strokes to increment and decrement the column or row allows the user to fine tune his selection as well as to efficiently compare between neighboring columns or rows.

3.2.2 Light Input

In our second small multiple, we display the set of lights that created the light transport matrix. In the example used in this paper, the set of lights used to light up the scene are the pixels from a projector. Therefore the structure of the set of lights is an image of the scene with the projector as the center of projection. This multiple serves two purposes; to show the light that is incident on the scene and the set of lights that constructed the light transport matrix shown in the main window. The current selection of lights used to construct the light transport matrix is displayed by using a bounding box that is layered over the data. If the display mode for the main window is 'Input', another distinct bounding box will be layered over the data to provide context to what the main window is displaying. As it is desired to relight the scene within the visualization to verify hypotheses, this multiple is also a canvas where the user can modify the intensity of the input lights using a paint brush. Any changes to the input light is not only displayed in the multiple but also propagated to the other multiple (observed scene or main window).

3.2.3 Observed Scene

There are two main functions for this third multiple; to display the observed scene under the current lighting input and the set of pixels that are used to construct the light transport matrix shown in the main window. The current selection of pixels used to construct the light transport matrix is displayed by a bounding box that is layered over the data. If the display mode for the main window is 'Output', another distinct bounding box will be layered over the data to provide context to what the main window is displaying.

3.3 Panel

The panel not only serves to remind the user which state the visualization tool is in, but it also allows the user to specify his queries. Radio and toggle buttons are used to let the user see the list of options that is available to him as well as which are the options are are currently selected or activated. In Fig. 4 we show our implementation of the panel which we are about to describe.

The panel can be divided into 3 main sections. The top most section allows the user to select a mode the main window will operate in as described in section 3.1. There are a few additional buttons and information available to the user. Beside the radio button for 'Light Transport', there is a update button that computes a new light transport matrix with the selected input lights and observed pixels. Beside the 'Columns' and 'Rows' radio buttons there are information panels along with a rewind and forward buttons by the sides. The information panel displays the current column or row that is being displayed in the main window. Since this information panel is specific to the mode, it is faded out when the relevant mode is not in use. The rewind and forward button allows the user to cycle through the columns or rows automatically in the main window in decreasing or increasing order. This will allow the user to quickly check if any input sample is corrupted or not. A toggle button labelled as 'Gamma' is the last button for the top section. This button allows the user to apply a gamma curve when displaying in all the modes except 'Light Transport'.

The mid panel controls the selection of input light and observed pixels as well as paint brush parameters. Radio buttons allow the user to switch between 2 modes of interaction for the 'Input Light' small multiple. When the radio button 'Paint Input' is selection, any clicks to the small multiple is registered as a brush stroke. If 'Select

Region' is selected, then mouse actions are registered for light input selection. A toggle button named 'Bind Selection' allows the user to use the same input light selection for the observed pixel selection. This will allow the user to create square light transport matrices where local interactions can be observed. There is also a set of radio buttons that control the shape of the brush. Brush size is selected by an Alphaslider. This allows the user to have a continuous range of brush sizes he can choose. Red, green and blue values for the paint brush are also selected by 3 separate Alphasliders with color coded knobs. A small box displays the current color selected for the paint brush. The color of the box is dynamically updated as the knobs are moved. This is to allow continual feedback to the user so that he can visually determine that the color is adequate while he uses the Alphaslider.

The bottom panel controls the display parameters for the all the windows. There are 3 toggle buttons that control each color channel. This allows the user to choose which combinations of the 3 color channels to display or squelch. The toggle button 'Fixed Ratio' determines if the separate color channels can be modulated and scaled differently. If it is turned on, all modifications to any color channel is propagated to the other channels. The toggle button 'Exclusion' determines the method that is used to map the color values 0 to 255 on the data. When it is turned off, the values 0 to 255 is remapped onto the selected color range. When it is turned on, the values that are selected in the color range are not excluded from the display. i.e. if the color range 50 to 100 is selected, then the other color values are automatically assigned to 0 while the values 50 to 100 is unaltered. 3 double knob Alphasliders are used for selecting the color ranges. Values between the 2 bounded arrows (knobs) in each Alphasilder represent the range of values that are currently being selected. The numerical values for the lowest and highest values selected are displayed at the respective ends of the Alphaslider. There is an additional Alphaslider that controls the overall amplification of the values in all the display windows. The current value of amplification is also displayed at the top of the corresponding Alphaslider. All the changes that are made in the Alphasliders are updated in real-time. This allows a continual display of data whereby the user can visually observe the effects of selecting different display parameters. This usually admits the user to form new insights about the data.

4 Methods

The visualization tool is built using OpenGL. OpenGL is chosen because it is very fast compared to other toolkits such as Protovis and it provides easy access to graphics hardware acceleration which is critical for displaying the amount of data in real-time. OpenGL is also very suitable because it allows multiple sub-windows to be displayed easily. Animations for zooms and pans are trivial with OpenGL as it can be done with simple camera manipulations. Layering of different objects and data can be easily represented in OpenGL by the depth of the elements in 3D space. More importantly, OpenGL is platform independent.

4.1 Viewports

Multiple windows are instantiated in OpenGL as multiple viewports. Each viewport will have its own context of controls and rendering parameters. This makes rendering multiple windows with different data easy in OpenGL. Each viewport has its own camera which allow for animations that are distinct from the other windows. An orthographic projection is used in our implementation because it allows different layers of data with the same coordinates to be layered easily.

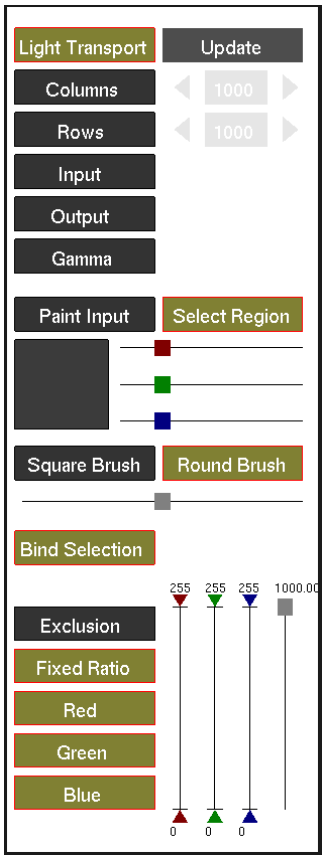


Figure 4: *Here we show a close up view of the panel. The top section allows the user to select the display mode for the main window. It is easy to determine which mode the main window is in from the highlighted toggle button. Notice that auxiliary options for the modes are faded out if the mode is not active. The middle section allows the user to select how the mouse actions are interpreted for the 'Light Input' multiple. There is a toggle button to allow the user to apply the same selection to the observed scene if he wishes. There are also Alphasliders where the user can use to adjust parameters for the paint brush. The small box on the left shows the current color that is selected for the brush. The bottom section allows the user to select display parameters for all the windows. This can be done using toggle buttons which control the channels and mode of remapping and Alphasliders which control the range selected.*

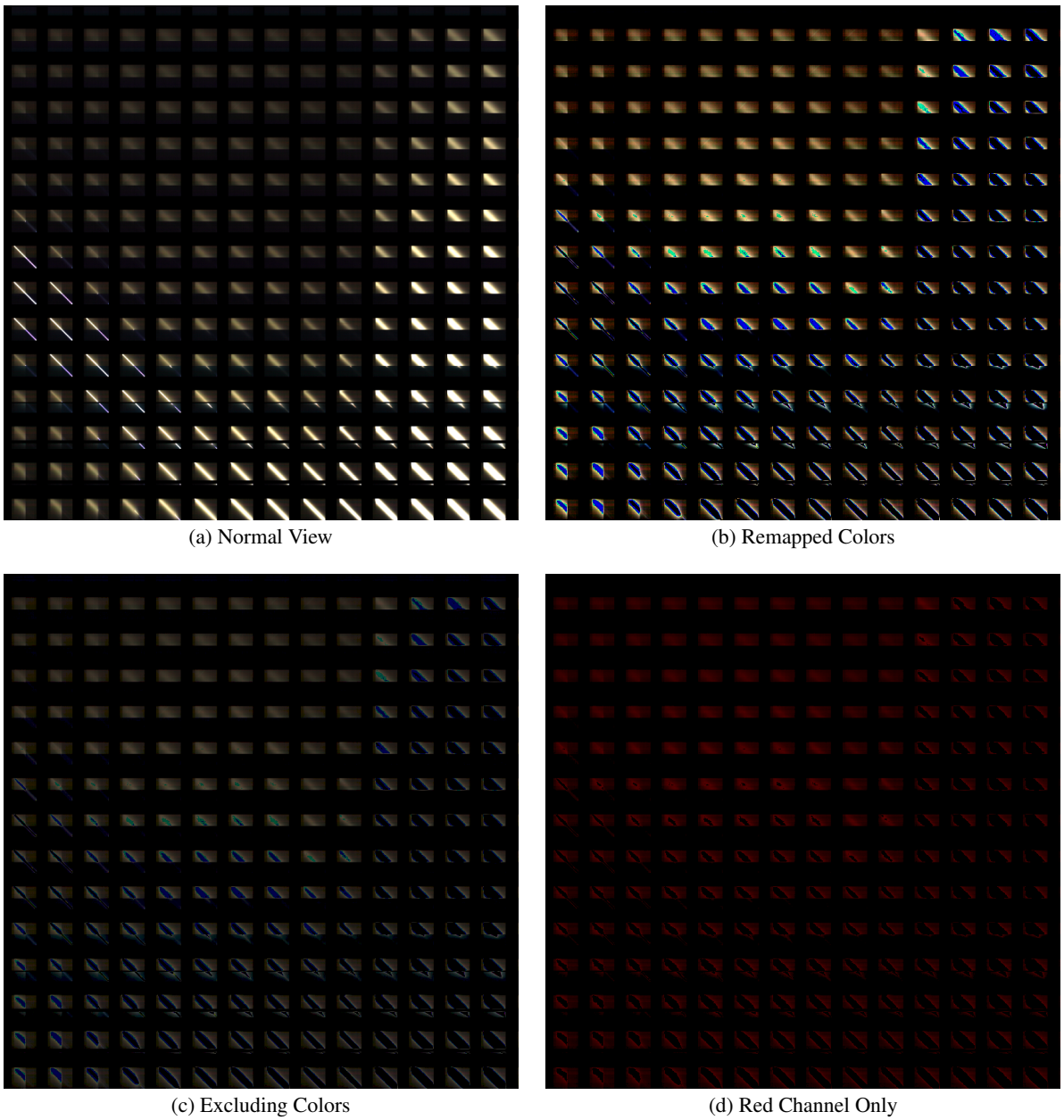
4.2 Ploygons

Each data point can be represented as a polygon in OpenGL. Using polygons is faster than drawing pixels onto the screen buffer because there is dedicated hardware for doing it and it admits for animation easily. We can construct an array of polygons arranged in a grid fashion resembling the arrangement of pixels to form an image. With an orthographic projection, polygons with the same coordinates will be mapped to the same point.

While it is tempting to use polygons for rendering all the data in all the viewports, it is not efficient even with hardware acceleration. The full light transport matrix has 16 million polygons and that takes approximately 1 second to render in OpenGL. As such, other rendering techniques like texture mapping will have to be used to achieve real-time interactive rates which is crucial in reducing the gulf of evaluation.

4.3 Texture

Texture mapping is technique in rendering that allows the addition of detail or surface texture onto polygons. This technique is heavily employed in computer games to add realism to meshes and applications such as Adobe Lightroom and Apple Aperture to efficiently display photograph libraries. Texture mapping is very fast because the graphics card has a hardware implementation for it. We can exploit this by storing the light transport matrix as an texture and mapping the texture onto 1 large polygon. This allows for real-time rendering for the light transport matrix. To achieve this, we compute the light transport matrix based on the selected input lights and observed pixels and generate a texture and load it onto the graphics



(a) Normal View

(b) Remapped Colors

(c) Excluding Colors

(d) Red Channel Only

Figure 5: *Here we show how the same data is displayed using different display parameters. (a) has the original data display with an amplification of 1000. (b) shows the same data with color values 0 to 255 mapped to the range 19 to 85. i.e. what was originally 19 and 85 is now mapped to 0 and 255 respectively; intermediate values are linearly interpolated. (c) shows the same data with color values outside 19 to 85 being mapped to 0. (d) shoes the same data with only the red channel.*

card. This process takes approximately 1 second. However, once this texture is uploaded, interaction with the light transport matrix will be in real-time.

There are several drawbacks for using this technique. Since each update to the light transport matrix takes 1 second to complete, interactively changing the light transport matrix becomes very slow. While this does not occur frequently, it is desired to interactively change the color values for each element in the light transport matrix when experimenting with different display parameters. Each texture that is used will have to be copied onto the graphic card's memory. A light transport matrix of size 4096 by 4096 takes up 50 megabytes of memory. Since each multiple window has its own instantiation of textures, the graphics card will run out of memory quickly and this reduces to speed of rendering to approximately 1 second per frame. Also, there is a maximum limit on the size of the texture. While this upper limit is not reached in our implementation, this is a major drawback for scalability as managing multi-resolution textures efficiently will be challenging.

4.4 Shaders

Shaders are a set of software instructions that are used primarily for rendering effects with a high degree of flexibility. In OpenGL, there are 3 kinds of shaders that are supported; vertex shaders, geometry shaders and fragment shaders. Vertex shaders allows for the transformation of each 3D point's position as well as texture coordinates. Geometry shaders allows for removal or addition of vertices. Fragment shaders allows for calculating the color of individual pixels. Since we are interested in real-time color remapping for the light transport matrix due to different display parameters, we can use fragment shaders to remove the need to reload a new texture every time the display parameters change.

We use 3 one-dimensional textures as inputs to the fragment shader for pixel color remapping. These one-dimensional textures function as look-up tables. They are each of length 256 and the values stored in it is the mapping for the value at the corresponding position. i.e. if the fifth element in the one-dimensional texture is 10, then the color value 5 will be mapped to 10. Since each color has its own one-dimensional texture, we can also color map each channel separately.

Since shading calculations are performed on the graphics hardware, it is very fast and efficient especially since it is highly parallelizable. The calculation of these 3 one-dimensional textures are very fast and since they are very small (256 bytes each), they can be uploaded to the graphics memory very quickly. This results in real-time color mapping for the texture when the display parameters are changed. An example on how changing display parameters for the same data is demonstrated in Fig. 5.

4.5 Putting it all together

To build the visualization tool, polygons, textures and shaders were used to render all the data in all display windows. The light transport matrix is rendered with a texture mapped onto a large polygon. For the small multiple, a smaller texture is calculated and used. Although the same texture data can be used, simultaneously having 2 extremely large textures in different context slows the rendering time drastically. Other data such as light input and observed pixels are rendered using polygons because they are sufficiently fast. While it is possible to use textures as well for these data, we did not want to use textures for all the windows as context switching can be slow. Fragments shaders are used to dynamically change the display values of the pixels in real-time as the Alphasilder is used. Radio buttons, toggle buttons and Alphasilder themselves are rendered using polygons. Text in the visualization tool is rendered using stoke fonts. Again, texture fonts are avoided so as to minimize context switching for textures.

5 Results

The visualization is built on a MacBook Pro 2.16 Ghz Core 2 Duo, 4 Gigabytes of RAM, ATI Radeon X1600 with 128 Megabytes of RAM using 1 thread with a screen resolution of 1440 by 850 pixels. It is able to achieve real-time interactively for zooming, panning, color remapping, painting, lights animation and region selection. Light transport matrix computation and updates takes 1 second but it is done only when the user decides to inspect another light transport matrix.

This visualization has been incorporated into the pipeline for our research. Collaborators are excited about the availability of the visualization and are looking forward to using it. It has thus far allowed us to quickly and accurately debug our light transport acquisition process. Through the visualization we have also discovered new phenomena that would have escaped us otherwise. For example, we have discovered that there are systematic biases in the camera sensor, producing bandings in the light transport matrix. While these values are very small, they accumulate via all the columns and can influence the result for the matrix inverse problem. Also, we observed systematic moire patterns that are present in all input images. While these patterns appear to be random noise within an input image, an animated sequence in the visualization tool will show that these patterns persists throughout most of the input images. We can also now visualize the effects of sub-surface scattering occurring in the scene. This is characterized by repeated structures in the light transport matrix. These effects are clearly shown in Fig. 6.

6 Future Work

While this is a big step towards creating a useful light transport matrix visualization, there are still many areas that can be improved. In this implementation the light sources are from a projector therefore allowing intuitive display formats for the input lights. Since light transport matrices are not limited to such light sources, it is interesting to think about how to display general light arrays.

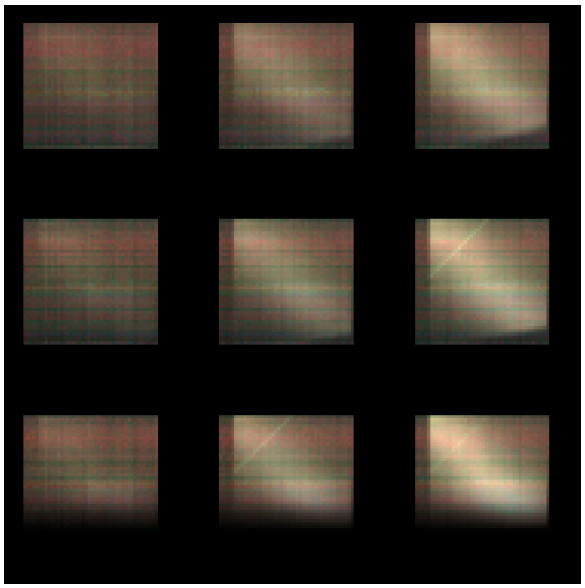
Currently, the small multiple 'Light Transport' is aliased especially if the light transport matrix is small. As such, better techniques for generating such previews will have to be implemented.

Since there is a maximum texture size for the graphics card, it is also challenging to allow light transport matrices that are larger than that. One approach would be to dynamically generate textures for regions of interest. This should also shorten the time to update the light transport as a much smaller texture will be needed to first show the overview of the matrix.

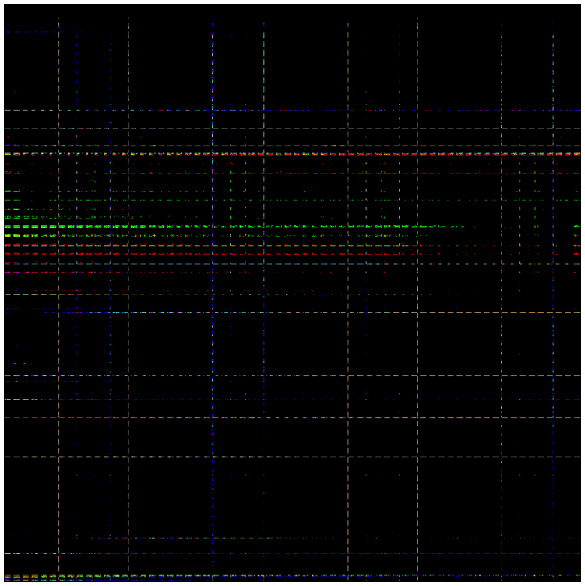
Tighter integration can be added so that the user can refer to input images or go to column or row mode for the element in question while in light transport mode.

Display parameters are currently implemented as linear scales. It will be useful to allow non-linear color mapping. This can be efficiently implemented as a curve GUI as per photoshop.

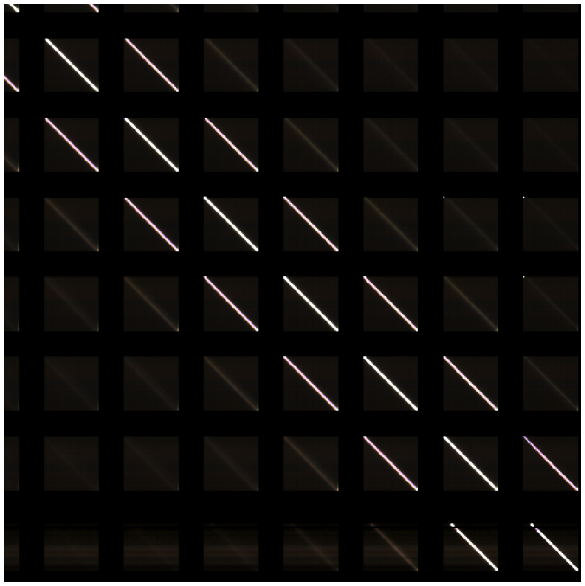
It will be also interesting to develop this into a tool with editing operations as well. For example, it will be useful to allow users to correct for systematic biases interactively in the visualization.



(a) Bandings



(b) Inconsistent exposures



(c) Subsurface Scattering

(d) Moire Patterns

Figure 6: *Here we show how our visualization elucidates certain phenomena. (a) shows strong banding across columns. This suggests that each pixel in the camera sensor produces distinct responses for approximately the same input. (b) shows the light transport matrix with values remapped into 0 to 2. We observe inconsistent exposures for different images and pixels. (c) shows strong sub-surface scattering effects in the scene where incoming light (diagonals) are manifested in neighborhood regions. (d) This is a typical input image that has moire patterns on the walls. While it may appear to be random noise, an animated sequence of inputs will show that these patterns persists through most of them.*

References

AHLBERG, C., AND SHNEIDERMAN, B. 1994. Visual information seeking: tight coupling of dynamic query filters with starfield displays. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, New York, NY, USA, 313–317.

NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. 2003. All-frequncy shadows using non-linear wavelet lighting approximation. In *ACM SIGGRAPH*, vol. 22.

NG, T.-T., PAHWA, R. S., BAI, J., QUEK, T. Q. S., AND HAN TAN, K. 2009. Radiometric compensation using stratified inverses. In *in Proceedings of IEEE International Conference in Computer Vision*.

NORMAN, D. A., AND DRAPER, S. W. 1986. *User Centered System Design; New Perspectives on Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.

PLAISANT, C., MUSHLIN, R., SNYDER, A., LI, J., HELLER, D., SHNEIDERMAN, B., AND COLORADO, K. P. 1998. Lifelines: Using visualization to enhance navigation and analysis of patient records. In *In Proceedings of the 1998 American Medical Informatic Association Annual Fall Symposium*, 76–80.

SHNEIDERMAN, B. 1996. The eyes have it: A task by data type taxonomy for information visualizations. *Visual Languages, IEEE Symposium on 0*, 336–343.

TUFTE, E. R. 1990. *Envisioning Information*, 4th printing ed. Graphics Press, May.

WILLS, G. J. 1995. Visual exploration of large structured datasets. *New Techniques and Trends in Statistics*, 237–246.