VRPN Imager

Russell M. Taylor II
David Marshburn
Ryan Schubert

July 16, 2008
Examples

• Run testimager_client
  – Run testimager_server
    • Show throttling
    • Show discarded frames
  – Run DirectX Video Server
    • #define vrpn_USE_DIRECTSHOW
Server Data Flow Diagram

- Imager(nx, ny)
- AddChannel(red, units, min, max)
- BeginFrame
- Region(red)
- Region(red)
- Region(green)
- Region(green)
- EndFrame
Client Data Flow Diagram

Description(nx, ny, nz, nchannels)

BeginFrame

Region(red)

Region(red)

Region(green)

Region(green)

...

SwapBuffers

EndFrame

Nano-CS meeting
VRPN Imager
Example Client: Setup and Run

vrpn_Imager_Remote *g_imager;  // Imager client object
g_imager = new vrpn_Imager_Remote(device_name);
g_imager->register_description_handler(NULL,
    handle_description_message);
g_imager->register_region_handler(g_imager,
    handle_region_change);
g_imager->register_discarded_frames_handler(NULL,
    handle_discarded_frames);
g_imager->register_end_frame_handler(g_imager,
    handle_end_of_frame);
while (1) g_imager->mainloop();
Example Client: Description

void VRPN_CALLBACK handle_description_message(void *, const struct timeval)
{
  // This method is different from other VRPN callbacks because it simply
  // reports that values have been filled in on the Imager_Remote class.
  // It does not report what the new values are, only the time at which
  // they changed.

  // Record that the dimensions are filled in. Fill in the globals needed
  // to store them.
  g_Xdim = g_imager->nCols();
  g_Ydim = g_imager->nRows();
}
void VRPN_CALLBACK handle_region_change(void *userdata, const vrpn_IMAGERREGIONCB info)
{
    const vrpn_Imager_Region *region=info.region;
    const vrpn_Imager_Remote *imager = (const vrpn_Imager_Remote *)userdata;

    // Copy pixels into the image buffer.
    // Flip the image over in Y so that the image coordinates
    // display correctly in OpenGL.
    if (strcmp(imager->channel(region->d_chanIndex)->name, "red") == 0) {
        region->decode_unscaled_region_using_base_pointer(g_image+0, 3, 3*g_Xdim, 0, g_Ydim, true);
    } else if (strcmp(imager->channel(region->d_chanIndex)->name, "green") == 0) {
        region->decode_unscaled_region_using_base_pointer(g_image+1, 3, 3*g_Xdim, 0, g_Ydim, true);
    } else if (strcmp(imager->channel(region->d_chanIndex)->name, "blue") == 0) {
        region->decode_unscaled_region_using_base_pointer(g_image+2, 3, 3*g_Xdim, 0, g_Ydim, true);
    }
}
vrpn_Imager_Region::
decode_unscaled_region_using_base_pointer()

// Bulk read routines to copy the whole region right into user
// structures as efficiently as possible.
Bool decode_unscaled_region_using_base_pointer(
    vrpn_uint8 *data, // Type and location of buffer
    vrpn_uint32 colStride, // Values to skip (3 for rgb)
    vrpn_uint32 rowStride, // Values to skip (3*width rgb)
    vrpn_uint32 depthStride = 0, // 0 means overwrite
    vrpn_uint16 nRows = 0, // Used to invert rows
    bool invert_rows = false, // Invert rows?
    unsigned repeat = 1 // 3 for mono → RGB buffer
) const;
vrpn_Imager_Channel

- vrpn_Imager_Channel(void) {
  - name[0] = '\0';
  - units[0] = '\0';
  - minVal = maxVal = 0.0;
  - scale = 1;
  - offset = 0;
  - d_compression = NONE;
- }

Avoiding Crashing

• Wait for the description before allocating or drawing regions
  – Image size not defined until then
  – Channels not defined until then
  – is_description_valid() can be used, or callback

• Check for description changing
  – Need to re-allocate buffers if new size
  – New channels added, old ones modified
Avoiding Tearing

• Also Avoiding FunkyChrome
• Wait for the end of a frame before drawing
  – Region buffers are per channel (one per color)
  – Regions are less than the full screen
Avoiding Flooding

g_imager->throttle_sender(count);
• Send only *count* more images, then stop
• Accumulates (not what you want with multiple clients)

g_imager->connectionPtr()->Jane_stop_this_crazy_thing(50);
• Imager doesn’t follow “The VRPN Way” and can flood
• Never return from the mainloop() call, data always ready when the handlers return
• Stops parsing after the packet with >= 50 total handled messages
Know What You’re Missing

- Register discarded-frames handler
- Nonzero value is known # missed
- Zero value is unknown # missed
- No callback is none missed
vrpn_Imager_Stream_Buffer

- public vrpn_Auxiliary_Logger_Server
- public vrpn_Imager_Server
- Multi-threaded full-rate logger
- Forwards “preview” as fast as possible
  - Separate throttle/loss for this
  - Actual lost frame counts still stored in file
- Can run on different host (bass) from server
Reading from a File

- **Standard VRPN**
  - Open device@file://filename, not device@host
  - Acquire pointer to playback control if needed
Region Types

- `uint8`
- `uint16`
- `u12in16`?
- `f32`
vrpn_Imager_Pose

- **get_origin(vrpn_float64 *xyz)**
  - Pixel (0,0,0) goes from half-pixel below to half-pixel above
- **get_dCol(vrpn_float64 *xyz)**
  - Vector from pixel (0,0,0) to one past last in x.
  - This is the total image width
- **get_dRow(...)**
- **get_dDepth(...)**
Not Yet Implemented

- Client request a subset of the image
- Client request a frame rate
- Client set exposure time
- Binning (value and client set)
- Full transcoding
- Compression
- ImagerPose not yet tested
## Parameters vs. Devices

<table>
<thead>
<tr>
<th>Parameter</th>
<th>digital camera</th>
<th>digital video camera</th>
<th>area scan camera</th>
<th>TEM</th>
<th>SEM</th>
<th>AFM</th>
<th>AFM simulator</th>
<th>Scientific camera</th>
</tr>
</thead>
<tbody>
<tr>
<td># of dimensions</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td># of pixels in dimension 1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of pixels in dimension 2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of images/image planes</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>type of data (height, intensity)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>native bit depth</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pixel format/interleaving</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>physical scale (pixel size)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>offset (and orientation) - read</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>offset (and orientation) - set</td>
<td>?</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>byte ordering</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>compression</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>binning</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inter-pixel delay</td>
<td>?</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>scan rate</td>
<td>?</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>frame rate</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>focus control?</td>
<td>?</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>