Table of Contents

1. Introduction

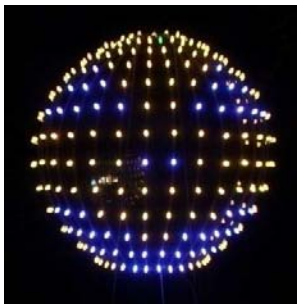
This past year I wanted to add some RGB pixel props to my display, so I thought I would try out the GE Color Effects (GECE) strings that seem to be quite popular. I chose a Snow Globe shape because it fits right in with "Christmas", even though it doesn't appear to be commonly used in Christmas displays. The spherical shape complements/contrasts other more typical prop shapes such as the cone-shaped Mega/Midi-Tree, so I thought that a sphere would enable some interesting new sequencing effects. Here are some examples:


Horizontal & Vertical Curtain effects

Spinning World

RGB sparkle

The spherical shape also allowed me to use the Snow Globe for other purposes. For example, as a New Years prop I could use it for a count-down, fireworks, and a scrolling text display. I was also able to start display setup and testing earlier than in past years by disguising it as a Halloween prop. ☺




Jackolantern and Ghost Faces

Fireworks


New Years scrolling text, count

Many other effects are possible with the Snow Globe – a few examples are visible within the 2011 eShepherds of Light videos on YouTube and Vimeo (search for "ESOL2011" DIYC)[1]. I think my favorite was the Spinning World animation[2], although the R/G/B sparkle[3] also looked cool (better in person than in the videos). Sorry about the poor video quality – I just made a quick set of videos with a cheap digital camera in order to show the sequences and the effects (I didn't even do that last year, so at least it's a small improvement ☹).

[1] YouTube playlist is at http://www.youtube.com/playlist?list=PLF58C3EDFF15916A9&feature=view_all
[2] In "Joy" http://vimeo.com/34711867, spinning world animation is shown at 1:22 - 1:28
[3] In "Wizards in Winter" http://vimeo.com/34712313, RGB sparkle effect is shown at 2:50 - 3:00

Since the Snow Globe is a relatively uncommon display prop but it enables many new and interesting effects, I thought I should write up what I did and share it with the DIYC community. However, this is just a description of what I did, and not a comprehensive "How To" guide, so your mileage may vary if you try out any of this info. I have tried to provide accurate info, and I can answer questions or provide more details about what I did, but some of these details may not be applicable or may need to be adapted for other displays or styles of sequencing.

Since this project involved several layers, first I'll start with some overview/background info, then describe the Snow Globe prop itself, then on to the controller (a retro-fitted Renard), and then the software and sequencing (briefly). Most of the sequencing and software details will be deferred to a separate article, because they fall under a broader software project from this past year related to effects generation and RGB pixel mapping in Vixen 2.x.

 All measurements are given in U.S. units unless otherwise noted.

2. Background Info

2.1. GECE Strings

The GECE strings appear to be available in several form factors: 25 ct vs. 36 ct vs. 50 ct, strings vs. icicles vs. snowflakes vs. snowmen, etc. From what I've read, they all use the same type of "bulb" (an RGB LED) and controller, but since I used 50 ct GECE strings, those are the ones that I will describe.

The 50 ct strings I used had 10 inch spacing and came with a controller and hand-held remote, and also some handy 2-part clips. The GECE nodes were similar in size and appearance to C9 bulbs; their size and spacing suggest that they are best suited for house/roof outlines or other fairly large props. I suppose they could also be more tightly packed onto a smaller prop, but that would make it rather bulky and bright (hmmm, maybe a 3D star).

2011 GECE Packaging


GECE Node


2-part Clips

I saw a lot of price variations. Early in the season I found 50 ct strings on the shelf at Costco for $65 USD, which works out to $1.30/pixel, shipped and including the power supply. This price point is better than some of the RGB pixels I've seen, and is not bad considering what all you get, but it is still more expensive than I wanted to spend for larger scale usage. So, I decided to try them out on a moderately sized prop first. I've also seen them in other online stores for significantly more than that price, and they went as low as $20 for 36 ct in early January (if you could find them), which is $0.56/pixel - a great price considering what you get.

2.2. Controller

The stock GECE controller has a number of hard-coded patterns, some of which look okay when free-running with a DIYC display. Since the GECE controller is at the head of the GECE string, it can be easily replaced with a DIYC controller and then Vixen can be used for full control over the display patterns. Of course, this will void your warranty, so removing the stock controller represents the "point of no return". I was a little nervous about that at first because these things aren't cheap, so I just started with one string in case it didn't work out. Fortunately the GECE protocol is very well documented[4,5], which makes the GECE strings well-suited for DIY hacking purposes. ☺

I won't repeat all the protocol details here, but basically each GECE command is a series of bits with some dead time between the commands. Each command contains an address to select the target node(s), followed by a brightness and color value (it's not full 24-bit RGB color but it's not bad for simple graphics). The GECE nodes self-assign an address based on the first command they see, and they don't pass that first command on to downstream nodes as they do for subsequent commands, so you can assign each node a unique address, the same address, or in groups if you want. Based on the bit timing, a 50 node GECE string can be fully updated at 25 frames per second (40 msec refresh), so there's no point in using a shorter frame interval in Vixen because the entire string can't be updated faster (unless you shorten the string).

There are a few DIYC controllers developed by other forum members to control the GECE strings, including a Prop controller[6] and also an Arduino[7], but since this is a DIY forum and I have all Renard-based controllers, I thought it would be fun to try to drive the GECE strings directly from a Renard controller. Some quick calculations showed that a PIC16F688 running at 18 MHz probably had enough bandwidth and memory to drive eight 50 ct GECE strings in parallel (bit-banging I/O pins to send out GECE commands), so this seemed like a very economical approach and a nice little DIY challenge. Since I already had a bunch of Renard-style controllers, it also meant that I did not need to order any new parts so I could jump in right away. ☺ Since Renard controllers are well documented in the DIYC wiki[8] and forums, I won't repeat all that info here.

As has been the case with some (most? all?) of my other DIYC projects, for this one I also made things more difficult than they needed to be by trying to incorporate some additional "features" at the same time. My reasoning was that I would have had a chance to experiment with some other interesting ideas along the way, even if the main project did not turn out well. So, my secondary goals for this project were:

- Use a Renard-based controller to talk to the GECE strings; so I called this thing a "Renard-RGB"
- Allow intermixing of GECE strings with other types of lights (LEDs or incandescent) within the same Renard controller or at least on the same COM port[9]
- To see if a PIC16F688 really could drive 8 GECE strings in parallel
- To start using a C compiler for firmware, and to check how inefficient the results are compared to ASM
- Figure out how to do simple RGB graphics in Vixen more efficiently

Since the resulting firmware eventually did what I intended it to (more or less), I guess you could call my transition over to C to be successful, although it was somewhat painful (more like annoying, actually). Things that I became accustomed to doing in MPASM with fancy macros were not allowed by the C compiler, so I had to use some awkward work- arounds. I didn't buy an expensive C compiler, and the free ones seem to be overly poor at generating efficient code. On the one hand, maybe it's not fair to expect a free or low-cost compiler to generate very efficient code, but OTOH giving away non-crippled tools as a way to sell more product has proven to be a successful business model by other corporations. Anyway, I finally found a low-

[4] At DIYC, http://doityourselfchristmas.com/forums/showthread.php?13062-RGB-LED-s-Now-Consumer-Grade-Hackable
[5] Elsewhere, http://www.deepdarc.com/2010/11/27/hacking-christmas-lights/
[6] http://doityourselfchristmas.com/forums/showthread.php?12444-A-DMX-controller-for-multiple-RGB-pixel-strings
[7] http://doityourselfchristmas.com/forums/showthread.php?17001-Late-2011-GE-Color-Effects-Arduino-thread
[8] http://www.doityourselfchristmas.com/wiki/index.php?title=Renard
[9] Intermixing concept used previously in Renard-HC project, http://downloads.eShepherdsOfLight.com/Renard-HC.pdf

cost C compiler that generated reasonably good code – the Boost C compiler[10] has economical hobbyist and standard licenses which fit my target price range. It is a little too restrictive on what you can put into inline ASM statements, but I was able to work around that with some effort. It looks like the CCX compiler[11] might also be a good choice – it is more flexible with regards to address placement and inline ASM, so maybe I will switch over to it at some point. However, it's also more expensive.

I was somewhat surprised at the source-to-binary code ratio for my firmware – it was about the same as my MPASM source code. I think this means that the heavy usage of MPASM macros in previous firmware gave approximately the same level of expressiveness as C, so it wasn't really worth the effort for me to switch over to C. However, having C code will be a little more portable if I move to another processor in future.

I noticed very occasional data loss on a down-stream PIC, but not the PIC immediately after the GECE strings, so I think this means that the GECE handling part of the firmware is correct and there's just a problem in the non-GECE part (I put some PWM code in there so I could use regular channels on the same COM port as the GECE strings). Since the GECE strings displayed the correct patterns, then I consider the 8 parallel strings goal to have been successful (although there are some limitations in this version of the firmware).

The technical details of the DIY controller I used with the GECE Snow Globe are as follows:

- Standard Renard BOM parts; any Renard controller can be used simply by re-flashing the PICs and connecting the I/O pins to the GECE strings (as described later), although I only tested one type
- Can run up to eight 50 ct GECE strings per PIC16F688 (8 x 50 = 400 RGB pixels = 1200 Vixen channels), up to a maximum of 16 PICs (6400 GECE nodes) per COM port, using a customized Renard-RGB protocol
- Non-GECE strings can be intermixed on same COM port or same controller; currently only PWM-style DC channels are supported (8 per PIC); all PICs on the same COM port must use the customized firmware
- For 50 ct strings the maximum frame rate from Vixen is 50 msec, due to the GECE protocol timing, not the Renard-RGB itself (50 nodes x 26 bits x 30 usec + 200 ≈ 40 msec to refresh an entire 50 ct string)
- The Renard-RGB plug-in works with Vixen 2.0.x and 2.5.x; 24-bit RGB colors from Vixen channel triplets are mapped to 20-bit GECE IBGR values by the Renard-RGB Vixen plug-in
- Lossless data compression on bi-tonal images; by eliminating redundant bytes from the data stream, 400 pixels require only 56 bytes[12], so 4000 GECE nodes can realistically be driven from one COM port at 115k
- The current version of firmware only supports bi-tonal images; any colors within the GECE color space can be used, but only 2 of them will be displayed (additional colors are mapped to the second palette entry)
- Auto-baud detection; 56k and 115k baud both seem to work okay if the PIC has an 18.432 MHz ext clock

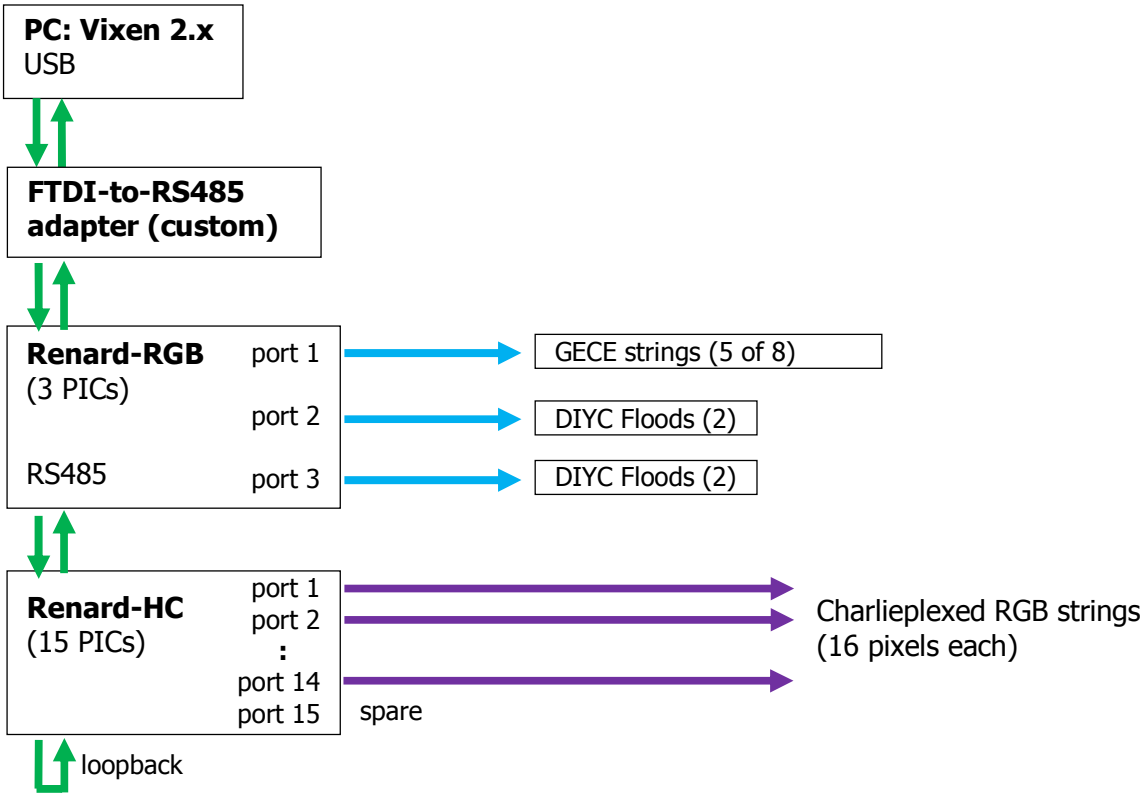
Although the protocol is designed to handle larger numbers of pixels, I only ran it with a few hundred nodes this past season (5 strings), so there are probably some bugs in there. ☹ But, I know that downstream PICs do receive data correctly so more than 8 strings might work. If anyone tries it on a larger number of pixels, let me know how it turns out (and please treat this as "experimental" software, not production-quality). ☺

Since I used only very simple graphics within my sequences, I only implemented bi-tonal graphics for now. This was due to time constraints rather than any technical limitations. I plan to remove that limitation in the next version, because I would like to run a little fancier animated graphics for next season.

[10] http://www.sourceboost.com/Products/BoostC/Overview.html
[11] http://www.bknd.com/cc5x/
[12] 2 x IBGR + 400 bits = 56 bytes; 115k at 50 msec refresh rate allows 576 bytes, which would be 10 x 400 nodes

2.3. Display Architecture

Below is a partial display architecture diagram showing how the GECE strings and Renard-RGB fit into my display:



Partial 2011 display architecture

The first PIC on the Renard-RGB ran the GECE strings, and other 2 PICs each ran a pair of DIYC Floods. That all worked quite well, except that I think I saw occasional drop-outs on the *second* pair of floods, which tells me there is likely a COM line problem or maybe a problem with the PWM code I used on the second PIC, not the GECE code on the first PIC.

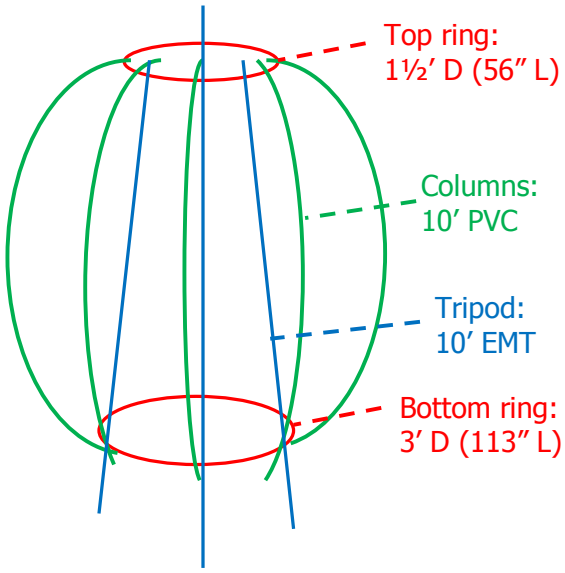
I intended to chain a Renard-HC after that, but I did not get the Renard-HC chipiplexing firmware merged in with the new Renard-RGB firmware due to time constraints - all PICs on the COM port need to understand the customized protocol. I hope to get that done for the 2012 display season.

The PC and FTDI-to-RS485 adapter were inside the garage for protection, and the Renard-RGB was out in the yard, close to the Snow Globe and Floods. The Renard-HC was on the garage door. It looks like I am starting to migrate away from the centralized controller architecture that I started with several years ago to more of a distributed controller architecture. I'll probably continue that trend as I add "smart props" into the display in coming years.

I used a return loop on the COM port so I could query performance/debug info from the Renards. It is very useful to be able to see some of the performance stats from the point of view of the Renard controller itself, and how that differs from the view within Vixen.

3. Snow Globe Construction

The Snow Globe structure was made out of PVC electrical conduit, metal conduit (EMT), and machine screws to hold it together. I fastened the GECE nodes onto the PVC sections to form vertical columns using 1 of the 2 clip parts from the GECE strings, and made top and bottom rings out of EMT for sturdiness. The rings were fastened to a tripod made of EMT, and then the PVC could be bent and held in shape (under tension) with only one fastening point at the top and bottom of each column.



Snow Globe structure

The Snow Globe structure was easy to build. I used the following parts:

- Five sections of 10 ft EMT (½ inch metal conduit)
- 20 sections of 10 ft gray conduit (½ inch PVC)
- 46 #8-32 x 2" machine screws and nuts, and 96 washers
- Five 50 ct GECE strings

First I cut 2 sections of EMT based on length calculations near the top and bottom of an 8 ft sphere, and then measured out bend marks (one per column) and used a pipe bender to form the rings. I didn't get a perfect circle, but it wasn't too far off. Since I'm not a welder, I drilled and hooked the ends together using ceiling hanger wire. I also drilled holes in the top and bottom rings at approximately 45° so the vertical columns could be fastened. To give fairly good coverage of the Snow Globe surface area, I decided to use 9" horizontal and vertical spacing. Since the GECE nodes have 10" spacing, this allowed a little slack between them vertically, and for consistency I also chose 9" horizontal spacing around the equator. This gave me 33½ columns so I rounded it down to 32 to make it a power of 2 and easy to divide into 360° (11¼° each).

Bend Marks

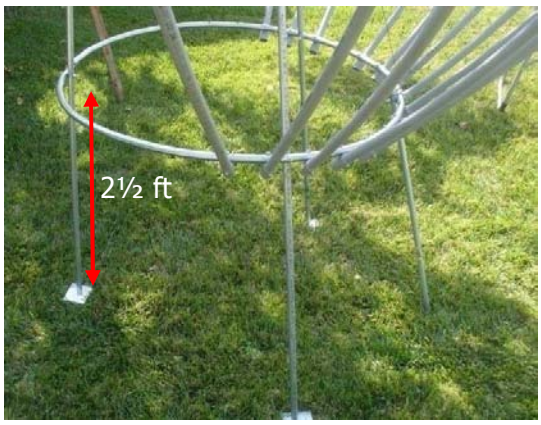
Pipe bender

Entire Ring Bent

Ends fastened

After drilling holes in the top and bottom rings, I used 3 more sections of EMT to form a supporting tripod shape, and then attached the upper and lower rings using machine bolts. This formed a rigid structure to attach the columns later. Based on the size of the sphere, there would be 2½ ft extra, so rather than cutting it off I left it intact and just raised the Snow Globe off the ground. I wasn't sure how waterproof the strings were, and we get a lot of dew and overnight moisture so raising it off the ground seemed like a good idea.

Tripod with rings attached



Raised off ground

I drilled a hole at each end of the PVC to attach them to the rings, and then additional holes every 9" to mount the GECE nodes. I wanted the nodes to line up so I measured, and I also used a nail in the end hole to check alignment as I drilled them. Then I inserted a roofing nail (large head) into the node holes and bent it over – later I hooked the GECE clips onto the nail heads to hold the nodes in place.

Measured drill marks

A nail as alignment guide

Insert + bend nails

Attach clips (later)

I rested the tripod on the ground while I attached the first few columns because I wasn't sure how it would turn out. I bolted one end of the PVC to one of the rings and then held the PVC in shape so it would line up with the ring at the other end and could be bolted there also. After the first few columns I stood the tripod up on its legs before attaching the remaining columns. I could have left it on the ground until the end and then raised it, but I didn't want to put much stress on the columns because I didn't tighten them until later.

Attaching columns to upper and lower rings

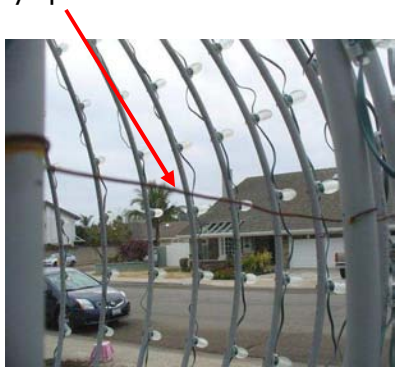
After all the columns were fastened to the rings and the basic Snow Globe structure was completed, I hooked the clips onto the nails and then snapped the GECE lights into the clips. I probably should have done that before attaching the columns to the rings in order to avoid awkward reaching on a ladder, but I put the lights on last because I didn't know if this process would work or not. The lights snapped into the clips very easily. I also added a ring of stiff wire mid-way down the columns to keep them evenly spaced.



Columns attached



Clips and lights attached


Wire for spacing

Since it would only be viewed from the front and sides, I only populated the front 2/3 of the Globe, which was 20 of the 32 columns. It looked okay this way; my fall-back plan in case it didn't was to rush out and get a few more GECE strings. (My controller could handle up to 8, for coverage of the entire 360° if needed).

Rather than cut the GECE strings, I zig-zagged them up and down adjacent columns; this pattern is evident in some of the test videos[13]. Using 9" vertical spacing between nodes (with 1" of slack) allowed 13 rows to just nicely fill within a 10 ft section of PVC. However, since the nodes are close together at the top of the sphere, I only populated every over node at the top, giving a half-row of nodes. The resulting 12½ nodes per column divides nicely into a 50 ct string, giving 4 columns per string without having to cut the GECE strings.

With only 3 EMT legs to support it, the structure was a little wobbly so I braced it with wire from the bottom front to the top back and pulled it tight to prevent the structure from sagging. I also added one more short leg in front since that area was taking a lot of the weight (only the front half of the Globe was populated) and I also ran guy wires from the top out to the sides to anchor it in place. This gave a fairly sturdy structure. We had a few days of high winds, but due to the open-air structure the wind just blew right through the Snow Globe with no harm. However, I did notice somewhat of a "golf ball" effect during the highest winds (the Snow Globe would twist a little around its vertical axis, like a golf ball as it flies through the air).

Since the Snow Globe structure was a first-time experiment for me, I did not give much consideration to storage afterward. Instead, I just dismantled the Snow Globe after the display season ended, removing the lights from the columns and the columns from the tripod. I figured that I would want to redo it again next time anyway, taking into account things that I learned from the first time around. If I stay with this size next time, I think the only change I would make is to enlarge the bottom ring and put it on the ground – this would leave more room for Nativity figures inside the Snow Globe, allowing it to be more like a real snow globe.

During my original planning/dreaming, I wanted to make a 30 ft Snow Globe, supported by some kind of geodesic dome structure[14] to encase the entire Nativity scene. The structure itself seemed doable, but it would take a lot of RGB pixels to cover that large of a surface area, and it would be expensive and require a lot of sequencing effort. So, I scaled back and settled on a smaller 8 ft Snow Globe – small enough to be doable as a first-time GECE project, but large enough to allow experimenting with simple graphics effects. I probably would never have finished if I tried for the larger one first. ☹

[13] Stock test sequence at http://www.youtube.com/watch?v=mOv-09CTx9w
[14] See http://www.desertdomes.com/dome3calc.html for a dome calculator.

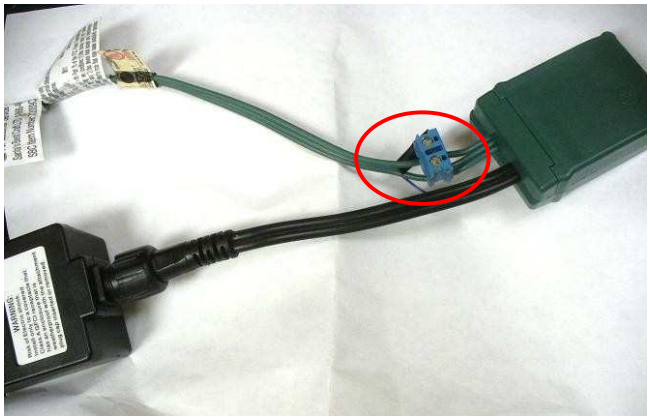
4. Renard-RGB Controller

In order to more easily fit into my existing Renard-based display architecture (and for a little DIY challenge ☺), I retro-fitted a Renard-style controller for use with the GECE strings. I used a Renard-HC PCB, but any Renard should work since the circuit and the core parts are the same. The "retrofit process" consisted of 2 steps:

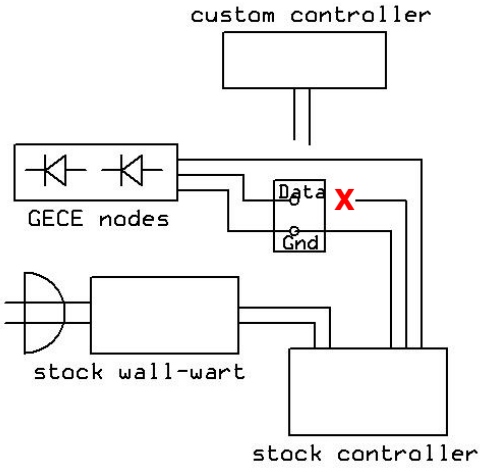
1. Re-flash the PICs with customized Renard-RGB firmware
2. Connect the PIC I/O pins to the GECE data lines

The controller I used had 3 PIC16F688s, so I set up the first PIC to run 8 GECE strings and I populated transistors on the I/O pins of the 2 remaining PICs so they could use PWM to control some DIYC Floods (equivalent to a pair of Ren4Floods[15]). All the PICs on the same COM port must understand the customized Renard-RGB protocol, so I flashed all 3 of the PICs with the new firmware.

Since I was going to use some custom firmware, I left the stock GECE controllers intact in case I didn't get the controller working in time. To allow "dual" operation, I only cut the middle (data) wire near the GECE controller (see the red "X" below), and then soldered a terminal block onto the data and ground wires. This way I could use the GECE wall warts for power and inject a custom data signal, or reconnect the original data wire and use the stock GECE controllers as well.



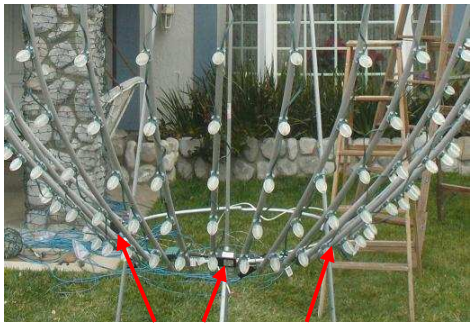
Terminal block added to stock controller



I attached the stock GECE wall-warts and controllers to the bottom ring of the Snow Globe using twist-ties, and then connected one data line and a ground from a cat5 cable to each terminal block. For weather-proofing, I just covered the controllers and wall-warts with plastic wrap or a plastic sandwich bag. Since I was using extension cords to the wall-warts, I paired them (the bottoms of the GECE strings were close enough to reach).

Inject data via terminal block

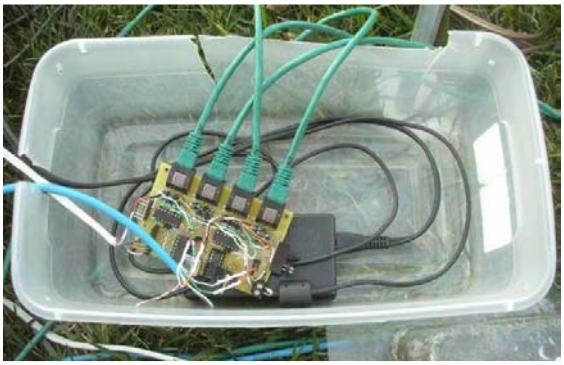
Plastic covering



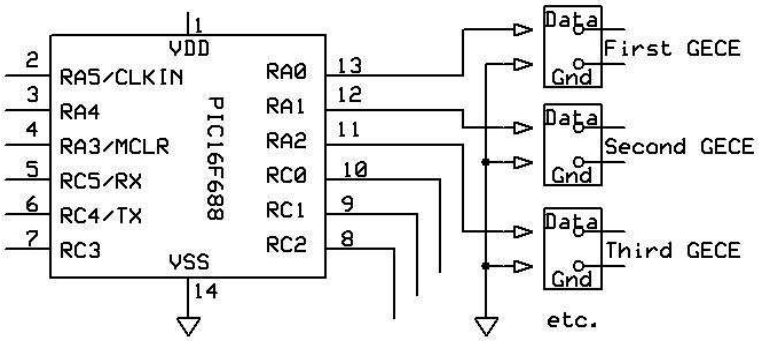
Wall-warts along bottom ring

[15] http://www.doityourselfchristmas.com/wiki/index.php?title=Ren4Flood

Since I was only using 5 GECE strings, I could fit all the data inside one cat5 by using a common ground. I made a custom cable to carry the data and common ground from the Renard to each of the GECE strings. This setup should work with any Renard-style controller, although I did not verify that.



Customized Renard-style controller



Cat5 with common ground to GECE strings

In the photo above, the green cat5 cables carry PWM transistor outputs to the DIYC Floods, the blue cable is incoming Renard data, and the white cable takes the GECE signals from one of the PICs out to the GECE strings as described above. Since I was using a laptop computer power supply for the DIYC Floods (shown underneath the Renard in the photo above), I ran the power through an onboard 5V regulator to power the Renard itself.

Limitations

When first powered up, the firmware waits 2 seconds and then initializes the GECE node addresses. Therefore the GECE nodes need to be powered up at the same time or before the Renard-RGB is powered up. I had the GECE wall-warts plugged into the same extension cord as the Renard's laptop supply, so this worked well for me but might not work so well in other setups. For example, if the GECE nodes are powered down, then the Renard-RGB must be also; otherwise the GECE node addresses will not be initialized again.

Since the controller was in the yard but the PC was in the garage, I used an RS485 line to send data to the Renard-RGB. This worked okay most of the time, but the startup sequence was a little finicky. For example, if I powered up the PC and RS485 line before the Renard-RGB and GECE strings, it would only get part way into the initialization sequence and then stop. I think the problem was that if the RS485 receiver is powered up there is enough voltage coming out of the EUSART to partially wake up the PIC, but not enough for it to do a clean startup. I've seen this happen with earlier Renard-HC projects also – after disconnecting power the PICs would remain alive until the serial line was also disconnected. I probably need to make the initialization logic in the firmware a little smarter to handle this case. Something to be aware of it you try out this stuff.

I had the GECE strings connected in a strange order so I hard-coded some channel reordering. After the show season, I turned that off, so for the released version the connections are as follows:

- RA.4 = first GECE string
- RA.2 = second GECE string
- RA.1 = third GECE string
- RA.0 = fourth GECE string
- RC.3 = fifth GECE string
- RC.2 = sixth GECE string
- RC.1 = seventh GECE string
- RC.0 = eighth GECE string

Note that the Renard-RGB was really only intended to be a convenient way for me to do some small-scale experimenting with GECE strings within my Renard-based display architecture, and not as a substitute for the fancier RGB pixel controllers that are available. Although the Renard-RGB protocol itself can theoretically handle up to 6400 GECE nodes, I only tested it with a few hundred nodes, enough for my purposes. The first 1 or 2 nodes sometimes "stick", so the firmware still needs a little timing adjustment I think.

5. Vixen Sequencing

Since I used a new protocol for the Renard-RGB firmware, I needed a new plug-in to go with it. Actually, I made 2 new plug-ins: an output plug-in to send out data in the format expected by the custom firmware, and a more general Vixen Effects Generator (FxGen) plug-in.

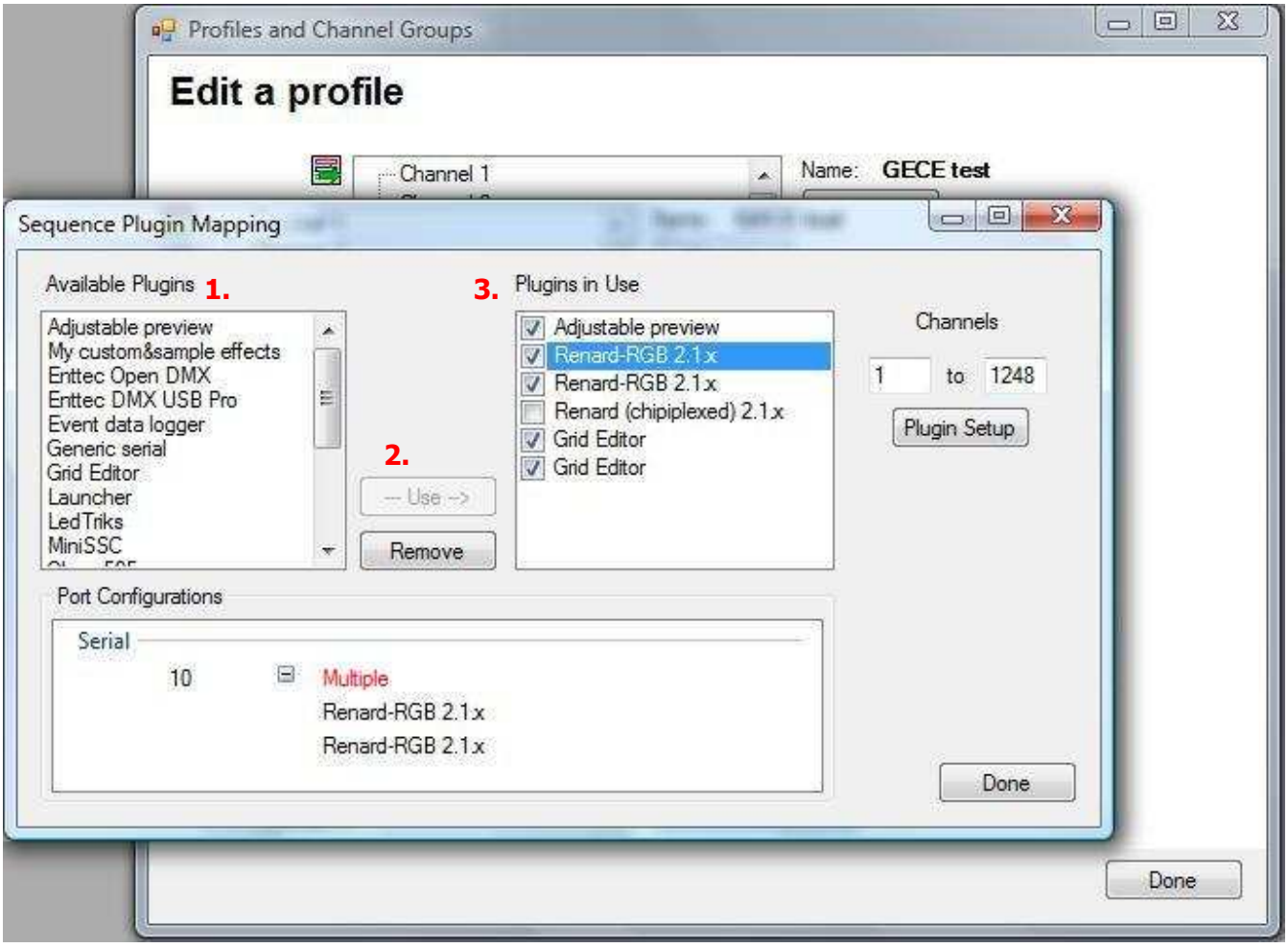
5.1. Renard-RGB plug-in

The Renard-RGB plug-in works similar to other Vixen 2.x output plug-ins: you assign a range of channels to it, and then it formats data bytes and sends them out over the COM port during Sequence playback.

During initial planning I forgot that Vixen plug-ins don't share COM ports well – they want to be the only one using it. However, I wanted to run 2 different types of props on the same COM port (Snow Globe and DIYC Floods), so I put some temporary logic into the Renard-RGB plug-in to make it behave like 2 separate plug-ins handling 2 different ranges of channels. Now I've replaced that logic with more generic "port sharing" logic, so the Vixen Profile or Sequence can have multiple copies of the Renard-RGB using the same COM port. I tested this feature briefly but it doesn't have a lot of mileage on it yet, so I might have broken something. Let me know if you try it and have problems.

Setup - General

Renard-RGB setup is similar to other Vixen 2.x output plug-ins. After the DLL is copied to the Plugins/Output folder (2.1 or 2.5 version is in the file name), then it can be added to a Sequence or Profile by selecting it from the Available Plugins list and clicking the Use button:

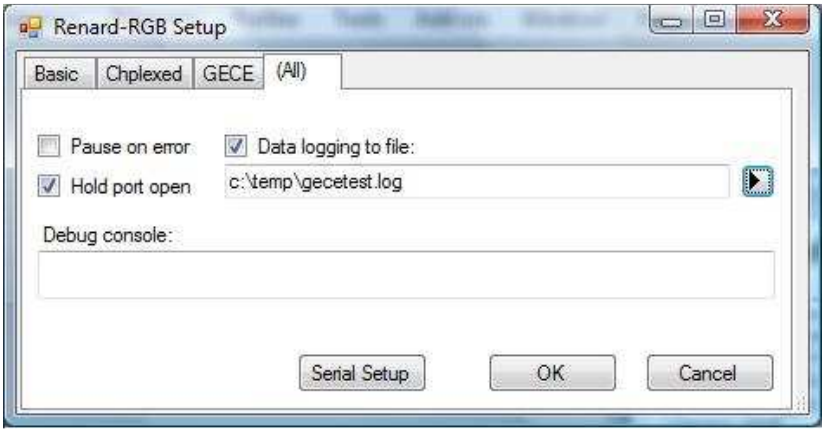


Adding the Renard-RGB plug-in to a Sequence or Profile

In order to run other Renard-style props on the same COM port, the Renard-RGB plug-in can be added multiple times as long as the channel range does not overlap with the other copies. Vixen will show "multiple" in red at the bottom, but this can be ignored since there is logic in the Renard-RGB plug-in to handle COM port sharing. The corresponding Renard controllers are assumed to be connected in increasing channel order so the Reorder function can't be applied across multiple instances of the Renard-RGB plug-in. I did some brief testing after adding the port sharing logic, but it might still have some bugs in it.

Each PIC running the Renard-RGB firmware can handle up to eight 50 ct GECE strings, which takes 8 x 50 x 3 = 1200 channels in Vixen. However, the 12½ node top-row mapping that I used on the sphere makes each 50 ct string actually looks like 4 x 13 = 52 nodes, so I added 8 x 52 x 3 = 1248 channels instead of 1200 for the sphere. Then I used the Grid Editor[16] to define a 32 x 13 RGB grid and Preview Bitmap on those channels. Only the first 20 x 13 of it actually had GECE nodes attached, but the plug-in doesn't know or care.

After assigning a channel range, I clicked on the Plugin Setup button to assign the run-time options. The popup Setup window displays one tab for each different type of RGB prop, and then a general tab for setting the port properties:



Renard-RGB general options tab

The "Pause on error" checkbox is useful for testing. If a run-time error occurs during Sequence playback, I want to know about it so I can take a look at what went wrong. When this checkbox is on, playback will be interrupted while the error is displayed, making it easy to notice if an error occurred. For live show playback, I turn this off and hope that any errors are not severe enough to crash. ☺

The "Hold port open" option is equivalent to a similar option in other plug-ins. This will leave the COM port open while the Sequence is playing, rather than opening and closing it each time. I always turn it on and haven't tested the plug-in with this option off, so I should probably hard-code it that way and then remove it from the Setup window.

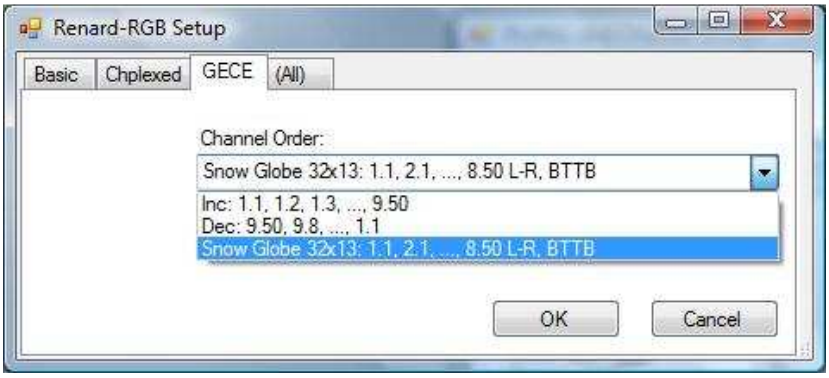
The "Data logging" options allow extensive debug info to be written to a file. I use this to check if the plug-in is sending out the correct byte stream to the controllers, or for other problem diagnosis. It generates a lot of data so I always turn it off for live show playback. The "Debug console" was meant to be a way to send or receive debug commands directly to the firmware, but I didn't ever implement that feature.

The Serial Setup button allows the COM port, baud rate, data, stop and parity bits to be selected, similar to other Vixen plug-ins.

[16] Described at http://doityourselfchristmas.com/forums/showthread.php?15152-How-i-did-Dumb-RGB-Pixel-Grid

Setup - GECE

The GECE tab allows the GECE handling options to be selected. It currently doesn't have much on it because I didn't really have many options that I needed to set:



GECE options tab

The Channel Order allows the ordering of the GECE nodes to be selected. The Inc and Dec options just address the GECE strings in linear order starting at the beginning or end of the strings, and the plug-in will use up to 1200 channels with these options. I had planned to use this option just to outline the house if the Snow Globe prop did not work out.

The Snow Globe 32x13 option is the one I used for the Snow Globe. It addresses the GECE strings in a left-to-right, bottom-to-top-to-bottom zig-zag order, and also turns on the 12½ row handling. This maps a 32 x 13 RGB grid in Vixen (1248 channels) into 8 GECE 50 ct strings (400 nodes) as follows:

Y\X	0	1	2	3	4	5	6	7	. . .	28	29	30	31
12	1.13	1.13	1.38	1.38	2.13	2.13	2.38	2.38	. . .	8.13	8.13	8.38	8.38
11	1.12	1.14	1.37	1.39	2.12	2.14	2.37	2.39	. . .	8.12	8.14	8.37	8.39
10	1.11	1.15	1.36	1.40	2.11	2.15	2.36	2.40	. . .	8.11	8.15	8.36	8.40
9	1.10	1.16	1.35	1.41	2.10	2.16	2.35	2.41	. . .	8.10	8.16	8.35	8.41
8	1.9	1.17	1.34	1.42	2.9	2.17	2.34	2.42	. . .	8.9	8.17	8.34	8.42
7	1.8	1.18	1.33	1.43	2.8	2.18	2.33	2.43	. . .	8.8	8.18	8.33	8.43
6	1.7	1.19	1.32	1.44	2.7	2.19	2.32	2.44	. . .	8.7	8.19	8.32	8.44
5	1.6	1.20	1.31	1.45	2.6	2.20	2.31	2.45	. . .	8.6	8.20	8.31	8.45
4	1.5	1.21	1.30	1.46	2.5	2.21	2.30	2.46	. . .	8.5	8.21	8.30	8.46
3	1.4	1.22	1.29	1.47	2.4	2.22	2.29	2.47	. . .	8.4	8.22	8.29	8.47
2	1.3	1.23	1.28	1.48	2.3	2.23	2.28	2.48	. . .	8.3	8.23	8.28	8.48
1	1.2	1.24	1.27	1.49	2.2	2.24	2.27	2.49	. . .	8.2	8.24	8.27	8.49
0	1.1	1.25	1.26	1.50	2.1	2.25	2.26	2.50	. . .	8.1	8.25	8.26	8.50

For example, in Vixen the RGB node at rectangular coordinates (2, 4) would be mapped to GECE string 1, node 30 (highlighted in red). Note that pairs of RGB pixels on the top row (Y = 12) map to the same GECE node – this allows each 13 x 4 rectangle to map to an uncut 50 ct GECE string. Of course this can all be changed around however you want by modifying the Renard-RGB plug-in; this is just the way I did it.

I just added the addressing orders that I needed so there are not many options. If anyone would like to use this plug-in with a different GECE node order, let me know and I'll try to add it.

Setup - Basic

I added a second copy of the Renard-RGB plug-in to handle 4 DIYC Floods. Since they were chained on the same COM port as the GECE strings, I selected the same COM port in the Serial Setup window. Then I used the Basic tab to set options for the DIYC Floods:

Basic/PWM options tab

The Basic tab is intended to be used with Renard PICs that drive one channel per I/O pin. I probably should have labeled it "1-to-1" or something like that. Since DIYC Floods contain 4 banks of LEDs and each bank is addressable as a separate channel in Vixen, I assigned 4 x 4 = 16 channels to the second Renard-RGB plug-in.

Currently I only have the PWM logic in the Renard-RGB firmware, because this is what I needed for the DIYC Floods. If I expand the firmware to handle non-PWM in future, then the PWM checkbox will allow me to select between PWM and non-PWM.

The Invert checkbox is a placeholder if I need to be able to select active-high instead of active-low in future.

Setup - Chplexed

I plan to merge my Renard-HC (chipiplexing) firmware into the Renard-RGB firmware (hopefully for the 2012 display season) so I defined a placeholder Chplexed tab to allow those options to be set. Currently it doesn't do anything.

5.2. Setting the Channels

After configuring the Renard-RGB plug-ins in Vixen, it was time to finally start turning cells on and off in Vixen. This can be done using any of the regular Vixen Sequence editing tools, but I chose a different route …

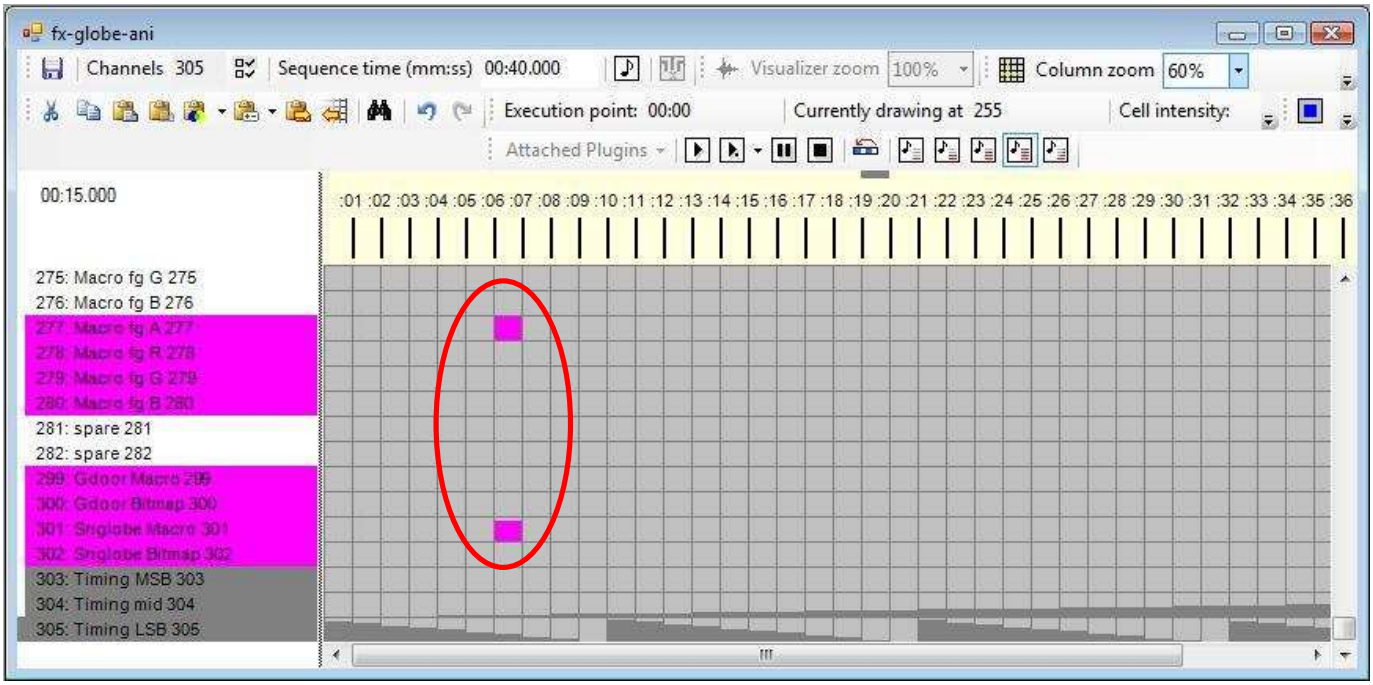
Last year I had added about 1600 channels to my Vixen 2.x Profile for a charlieplexed RGB grid[17]. Vixen can easily handle that many channels, but it made the Vixen sequence files much larger, and setting all those channels was rather cumbersome. This year, instead of adding yet *another* 1200 channels and then having an even larger Profile and more cell editing work, I back-tracked and moved all those channels into a separate "virtual" Profile for my RGB props. Then I made an Effects Generator (FxGen) plug-in that would instantiate those RGB plug-ins and dynamically generate channel values for all those RGB channels, so they could be sequenced using only a few mouse clicks. This kept my main Sequence files fairly compact and more manageable.

FxGen is somewhat like VMWare - I added FxGen to my main Profile, and then added the Renard-RGB plug-ins as part of the virtual RGB Profile rather than my main Vixen Profile. This kept my Sequence files and main Profile much smaller than last year - about 1/10 the size that they would have been (300 channels instead of 3000).

[17] http://downloads.eshepherdsoflight.com/Howidid-DumbRGBPixelGrid.pdf

Most of the effects that I ran on the Snow Globe were written as RGB "macro functions" in FxGen rather than setting individual cells by hand within the Vixen sequence itself – although that also would have worked, because the plug-ins don't know the difference. For each RGB prop I defined one channel to control the Macro function itself, and then a few more channels for the Macro function parameters (pink channels in the screen shot below). For example, for the Spinning World effect, I defined Macro function code 235 to draw the spinning world bitmap on the target plug-in's channels, with another Parameter value to control the speed and direction of the spin. Many of the Macro functions also use 4 channels to define a [A, R, G, B] color value for the effect.

So then whenever I wanted to play the Spinning World effect, I simply set the Snow Globe's Macro function channel to 235 and the speed channel to how fast I wanted it to spin. I did this using the regular Vixen cell editing functions, but since it was only a few cells, this was very easy to do:



FxGen channel cells to trigger Spinning World effect

Since the FxGen effects are represented by a small number of real Vixen channels, they can be precisely synced with the rest of the lights, and yet they can be easily added into other sequences or applied to other types of props. Some of the effects are actually animated bitmaps (such as the Spinning World), created using Windows Paint, and all of them use regular GDI-based drawing functions (mapped to a bunch of Vixen channels), so I think that my goal of simpler Vixen RGB graphics was also somewhat successful (at least for me).

I'm trying to keep this article down to a reasonable size and FxGen is a whole other topic, so I will need to defer the rest of the details to another article since they are not directly related to the Snow Globe or the RGB-enabled Renard controller.

6. Future Plans

I didn't get quite as far along with all of this as I would have liked, but I am reasonably happy with the results so far and it was rather fun stuff to play with, so I think I will develop it further for the 2012 display season. As stated earlier, the Renard-RGB is not meant to replace any of the other GECE-capable DIYC controllers that are already available. I see this project as more of an "experimental stepping stone" for myself or anyone else who would like to use it with a smaller number of GECE nodes within an existing Renard-based layout, prior to committing to building a more powerful RGB node controller.

At the moment my plans for the 2012 display season are to add some other smaller RGB grids (~ 16 x 16 pixels) into my display, and I'd also like to run a little fancier graphics (mainly animated bitmaps). Therefore, I will probably remove the palette size restrictions in the Renard-RGB firmware first, and then try to get some additional effects added to FxGen. For the time being, I will be staying on a Vixen 2.x code base, but if anyone wants to use this with Vixen 3.x it should be fairly easy to make a "compatibility layer", since the bytes have to go out over the wire the same way, regardless of which Vixen is driving them.

I'd also like to finish converting the Renard-HC firmware to C and merge it in with this new Renard-RGB firmware, so I only have one version of firmware but I can use it for multiple purposes (GECE, other RGB props, or AC SSRs). This would also allow me to put my earlier charlieplexed RGB grid onto the same COM port as the GECE strings – I'd like to see if that will actually max out a COM port at 115k baud (I haven't had enough channels to do that yet).

I can't guarantee anything at this point since my priorities are subject to change, but if I do make those changes I'll post the resulting firmware and plug-ins so it's available to anyone who wants to dabble with low-volume GECE nodes in a Renard-based display. Conversely, I ask that if anyone finds bugs or makes enhancements that you let me know, so I can update my copy of the source code.

If anyone would like to try out this stuff but needs a new feature, let me know and I'll see if I can add it.

7. More Information

The Renard-RGB plug-in and firmware are included in the files associated with this document. The FxGen plug-in will be included in a separate FxGen write-up, since it needs some clean-up prior to general release.

If you have any questions or comments for improvement of this article, please send an email to techguy@eShepherdsOfLight.com.

As I worked on this project, I found it to be an interesting illustration of God's character. First, there is the obvious analogy that Jesus is the Light of the World, so turning lights on and off in the dark is obviously representative of God shining His Light on all of us. But then there is also an interesting illustration about God's triune nature from RGB LEDs. An RGB LED is a single device, yet it has 3 very distinct parts or behaviors (red, green, and blue) that each serve separate purposes. This seems to be similar to God himself – one God, but 3 Persons: the Father, the Son (Jesus), and the Holy Spirit. The RGB colors may be somewhat representative of this also – green for the Father (green representing Creator of our world), red for His Son Jesus (who shed His blood for us on the Cross), and blue for the less-visible but ever-present Holy Spirit. I suppose there are other analogies that could also be made, but I'll leave those as an exercise for the reader.

License

Disclaimer

This project is a work in progress, not a finished product, so any of its components are highly experimental in nature. I got it to work okay for what I was doing, but it has not been tested thoroughly under all possible conditions, and some features probably have not been tested at all. This should be taken into consideration if you will be trying out any of this stuff. However, please feel free to ask questions or make comments on the info in this article; I tried to cover too much info, and as a result probably made some mistakes along the way.

Revision History

Revision	Date	Description
1.0	2/4/12	Initial version written up and released

-eof-