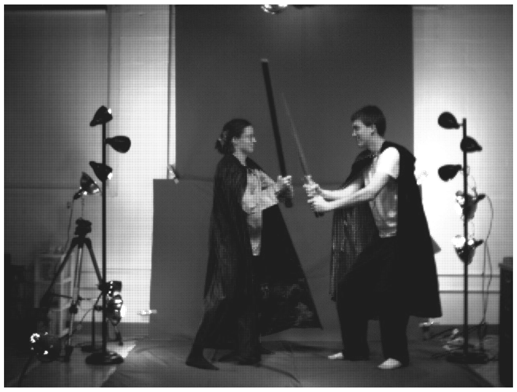
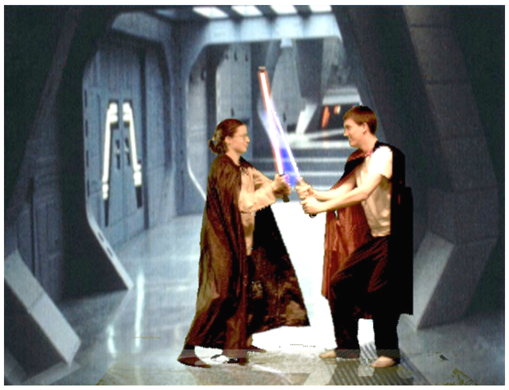
CS371 Spring 2007 — Assigned: Mon. 3/5/07
Prof. McGuire — Due: Mon. 3/12/07 8:00 pm

Project 0x04: FX


Before


After[1]

This week you will create digital effects similar to those used to produce the Star Wars and Matrix movies. The images above show the raw data captured on a film set and the final composited image produced from the program that you will implement. Note the new background, reflections of the actors, glowing light sabers, and correct gamma adjustment.

This is not a cumulative assignment, so you will not be able to reuse much code from previous weeks and will not re-use this code in future weeks unless you choose to a final project based on film effects. This gives you some liberty to make your code relatively specific to the input data, however you should still exercise good software design judgment in structuring and documenting your code.

The documentation requirements for this week's project are more detailed than for previous weeks because you need to demonstrate the intermediate results of your computations. Read them carefully before you begin!

You may work with one other person on this project. Many steps of the video editing process can be made easier if you are skilled in the use of software like Photoshop and Final Cut Pro. Consider forming your team to balance art and programming skills.

[1] Special thanks to Kathleen Creel, Daniel Fast, Scott Olesen, and Kate Foster for helping to create the video input sequences for this project.

FX Specification

Implement a digital effects pipeline for movie frames that performs matting, compositing, and bloom on specific props.

1. Devise and implement parsing of a human readable (text) configuration file format that specifies:
    a. the name of a series of input images,
    b. the name of a single or series of background images,
    c. the name of a series of output images, and
    d. any constants that will be needed by your program
2. Read the name of the configuration file from the command line arguments to your program. I.e., you should not need to recompile your program to process a different input sequence.
3. Load monochrome .PNG files with sequential filenames
4. Perform nearest neighbor Bayer demosaicing
    a. You MAY NOT use the G3D Bayer functions or look at their source code
    b. You MAY NOT use Bayer demosaicing code found on the internet
    c. You may use any other part of G3D
5. Crop the input image to the specified bounds or otherwise remove garbage from the original film set
6. Use green-screen matting to separate the actor from the background in the input sequence
7. For the light-saber sequences, make the light sabers appear to glow by applying a bloom-like effect only to them
8. Composite the results against a new background (or series of background images)
9. Gamma correct the final image
10. Write the results out to a series of images.
11. Create an MPG, MOV, or AVI file that can be played on the Macintosh in the Unix lab. The movie will be worth substantially more points than the other steps when graded. You do not have to automate this step.

If you choose to add additional non-programmatic effects during the editing of your movie, you must submit a separate movie that contains only effects produce by your program.

You will find many Photoshop tutorials on the web that describe the light saber effect and green (or blue) screen matting. These can be very helpful in designing your algorithm. However, excepting extra credit, your final implementation must be purely algorithmic. For example, you cannot manually select the light saber in each frame as part of your effects pipeline.

You may choose to use any language, platform, and libraries you wish, although only C++ and G3D on the department's FreeBSD systems are supported. If you use C++ as I recommend, please use Java (yes, Java) coding conventions for your C++ code.

Implementation Tips

You can find several video sequences at /usr/local/371/data/video. Sample Bayer before and after frames are in /usr/local/371/data/video/bayer.

Unless you are implementing some form of temporal coherence (for extra credit), there is no need to have more than one frame of animation loaded at once. Set up a loop to load each image, process it, and then move on in order to save memory.

The original Blue Screen Matting paper, which described the technique that had already been in use for about 20 years is at:

> http://www.cc.gatech.edu/dvfx/readings/smith-blinn-s96.pdf

You don't have to implement Vlahos' method exactly. Use whatever it takes to algorithmically distinguish between the green backdrop and the actors in the foreground. I've found that the following tricks are often necessary:

> Since we have a green screen, swap green and blue in Vlahos' equation. Then replace the "blue" with max(red, blue).
>
> Always set bright (> 0.95) and dark (< 0.01) pixels to alpha = 1.0 regardless of what the equation computes.
>
> After computing alpha, increase its contrast with alpha = (alpha – k_3) * k_4
>
> Tune the constants interactively—in onUserInput make different keys to change different constants up and down, so that you can watch their changes in real time instead of recompiling your whole program.
>
> Multiply the final alpha channel by a hand-drawn "garbage matte" that is black in areas that you always want to cut out and white in areas that may contain foreground objects. Because the camera is stationary relative to the set, you can paint a single garbage matte for the whole video sequence.

The basic idea behind making a light saber glow is to first create a matte for the saber (which is the opposite of the process of creating a matte for the actor against a green screen!), blur out the saber, and then add the blurred image back into the original. For a more authentic appearance, combine a colored blur with a white core, as shown on these websites:

> http://r2.robotbuilders.net/cgi/DaiBuckley/index.htm
>
> http://www.alienryderflex.com/rotoscope/

Even if you have trouble with one of the effects, that should not prevent you from implementing later stages of the pipeline. Use Photoshop or the Gimp to manually implement that effect (or produce appropriate input from a later stage from scratch) so that you can get points for most of the requirements.

I gamma-correct after applying all other effects. But you can probably make it work in any order after Bayer; the light saber is a hack anyway.

Sometimes the data just isn't there in the input. If you can't find a good set of matting constants for a data set, maybe that's a bad video sequence. Try another one, or hand-draw a good sample image to start with when you're debugging your algorithms. Tweaking the constants for matting is a time consuming and frustrating step, because there are so many of them and it is hard to get the foreground separated well.

There are several ways of turning your frames into a movie. iMovie and Final Cut Pro are both available to you in the Unix and Very Special Purpose labs, and the OIT staff can help you with other methods.

Innovate!

Some of the video sequences include the use of a blaster rifle, which shoots at the Jedi (he then deflects the blasts back at his attacker using his light saber). You can add blaster shots by manually drawing them for each frame. That's time consuming and may produce poor results, however. A better method is to write code that lights up pixels (like your light saber effect) along the path of the blaster shot programmatically. You'll have to specify the shot start and end points in your text file, of course.

Nearest neighbor demosaicing produces relatively poor results. This means that your matting algorithm will likely have bad results around the edges of objects and that the images will look either slightly grainy or blurry. There are several better methods for demosaicing. Bilinear demosaicing interpolates the missing channels from the eight neighbors at a pixel rather than just using one of them. Malvar et al.'s method performs more advanced filtering across a larger set of pixels.

It is common to composite multiple layers of effects. For example, a film might contain a 3D rendered background, live action actors, and then weather effects like rain and smoke composited over the actors.

Manual retouching is a fact of life in film production. Green screen mattes are regularly retouched in photoshop to fix locations where the algorithm was in error. In many cases, the wires holding up flying actors and space ships are manually edited out as well. You can retouch your frames and intermediate results for the movie (e.g., manually fixing matting errors) as long as you also produce a video that is unmodified.

Behind-the-scenes videos are a great way to demonstrate your work and will receive extra credit. Instead of just a final result video, submit a video that shows the following sequences in order:

1. Final result
2. Input
3. after Bayer
4. after Gamma
5. after Matting
6. after Compositing
7. after Bloom
8. after other Effects (final result again)

There is great potential for artistic extra credit on this assignment. Here are just a few examples:

You can implement filters of your own design, e.g., a median filter and black outlines around silhouettes will make the foreground look like a cartoon; you can then composite it over a cartoon background.

Create a number of different sequences and edit them together to tell a story. Add dialogue, sound effects, and music.

Composite multiple "foreground" characters into a single scene. Characters don't have to appear on screen in their original position—you can slide them and the background to make them walk or to match virtual camera movement.

Flipping an image upside down and adding a faint version of it back into the original simulates a reflective floor.

Implement artificial depth-of-field effects to defocus the foreground or background. Changing focus between foreground and background (a rack focus effect) is a good way of shifting the viewer's attention.

Although you can only work with at most one other student on the assignment proper, you can carefully plan with another team to each produce different effects for different scenes and then combine your scenes into one extra-credit movie.

Submitting Your Solution

1. Put your name, e-mail address, and the name of the file in a doxygen comment at the top of `each file`.

2. Create a “doc-files” directory that contains a **readme.html** file with your name, e-mail address, partner’s name (if you worked in a pair) and anything you’d like to point out to me when I’m reviewing your program.
    a. If there are known bugs, extra credit features, or design points of note, list them here.
    b. Credit any code that you found on the internet.
    c. If you are using your 2-day grace period or a prearranged deadline extension, explain that in the readme file.
    d. Explain the motivation for the algorithms that you developed.
    e. You <u>must</u> present an image sequence showing the initial input and the output **of each image processing stage** applied to the same frame, to prove that your algorithm is working correctly. This is the first place I’ll look when grading.

3. Put your completed movies in the doc-files directory. Make sure that they play on the Mac in the Unix lab. Make sure your readme.html file explains what each movie represents. Movies should be compressed down to fewer than 8 MB each.

4. Delete all generated files using “icompile --clean”. Do not hand in a build this week.

5. Change to the **parent** directory of your project and run the FreeBSD command:

```
submit371 fx
```