Project 0x05: FX
Due Friday, Oct. 31, 2008 at 6:00 pm

1 Specification

Implement a digital pipeline for rendering cinematic effects and produce:

1. A still-frame from the home-made “Jedi” film by:
   a. Loading one of the images inside from data/video/jedi.zip
   b. Demosaicing to produce a color image
   c. Removing the background outside the green screen using a garbage matte
   d. Removing the green screen using green screen matting
   e. Detecting the light-saber and making it glow by:
      i. Replacing light-saber pixels with a solid color
      ii. Blurring that solid color
      iii. Overlaying a white core at the location of the original light saber
   f. Compositing into an appropriate scene

2. A movie from the actual “Matrix” film by:
   a. Loading one of the foreground sequences from data/video/matrix/
   b. Performing green-screen matting to recover foreground and alpha
   c. Loading one of the background sequences data/video/matrix/
   d. Manually finding the timing offset, scaling, and cropping needed to align the foreground and background in time and space
   e. Compositing the foreground and background
   f. Saving the output sequence

3. A results page that clearly demonstrates each stage of your algorithm working (as stills and optionally as video), describes how you implemented, and describes any bugs or additional features. Think about how the “Matrix: Behind the Scenes” film we saw in class showed before/after/matte shots and had videos blending between them in slow motion to explain how the effects were built.
I included an entire sequence of frames for Jedi, but you are only required to process a single frame (and you can shop around for the frame that works best with your implementation). Likewise, I included two Matrix green-screen sequences but you are only required to use one.

All of the data has been “cleaned up” from the raw camera input, but beware that it still contains some real-world errors, misalignments, and artifacts that you have to work around.

1.1 Restrictions

You may not use any of the G3D Bayer demosaicing methods in your solution, nor may you look at their source code. You must write your own demosaicing, green-screen matting, and compositing methods. You can share any other code (e.g., GUI, constants, helper functions, extra features, Gaussian blur code) between groups and can help each other with video editing. **Just be sure to clearly credit other’s work, as always.**

**Do not use the video recorder built into the G3D GUI to output your video or stills.** Instead, explicitly instantiate a VideoOutput object for video and Image3uint8, Image3, or GImage for stills.

1.2 Evaluation

Your work will be evaluated for the following properties, in order of significance:

1. Satisfaction of the specification
2. **Mathematical correctness and quality of mattes**
   Tip: you should have nice fractional alpha values along the borders of objects, and foreground colors that are compensated for green bleeding in along the edge. Don’t just knock out “green” pixels and leave the others alone.
3. Readability, including documentation
4. Presentation of results
5. Design (including performance)
6. Programmatic correctness (compiling, memory management, safety)

In the event that you have known bugs, add a documentation section explaining what they are, how you attempted to debug them, and what you think the problem is. Use images as necessary. In general I will not deduct many points for bugs that you are aware of and investigated.

2 Working with Video Files

You’ll want a way to view input and output video on FreeBSD. There are two tools available to you. /usr/local/37/bin/viewer/viewer is the G3D viewer application. You can run it by typing that whole thing or changing your PATH variable in ~/.local_bashrc (see the handout from day 1) to be:

```
PATH=$PATH:/usr/local/37/bin:/usr/local/371/bin/viewer
```

Another tool on FreeBSD is mplayer, which is the Unix version of Windows Media Player. It has a nicer interface interface but produces lower-quality playback than G3D.
You may only check in video results to SVN that are smaller than 7 MB. Please don’t check in more than three videos. Note that raw AVI output will be huge and way over this limit. You can either output raw AVI or high-bitrate MP4 and use QuickTime or iMovie on the Macs in Jessup to edit and recompress your results, or output moderate bitrate (e.g., 1500 kps) MP4 files directly. Use a lot of stills in your results (as JPG, please) to augment any video results.

3 Discussion

This is the last regular project. It is designed to help you transition from projects that I designed with a clear idea of how to accomplish them to one that you design and for which I don’t know the best way to implement it. This is a relatively short project, but it also contains less guidance. Be aware that you’re going to have to spend more time than usual figuring out how to do each step and reading documentation. Actually implementing the steps once you’ve figured out the algorithm shouldn’t be too hard, although you’ll find that there will be lots of magic constants and scene-specific tricks that have to be manually adjusted.

Some of the things you’ll have to figure out are:

- How to adapt Blinn and Smith’s “Blue Screen Matting” 1996 SIGGRAPH paper to implement green screen matting.
- How to paint and import a garbage matte
- A design for the program
- Which of the four Bayer conventions was used for the Jedi shot
- A useful GUI for working with video frames
- How to present your work in a clear and compelling manner—including the physical process of moving video to a place you can title and edit it, and where to use transitions, slow-mo, etc. to show off your results.

You know how to work with data on both the GPU and the CPU. Working on the GPU allows your program to run much faster, which means that for video sequences you can see the results in real-time (or faster) while debugging. But you’ve also seen that it is very hard to debug shaders because error messages are frequently useless, mistakes will crash your computer, and there’s no way to get anything other than a single color out of the shader. You’ll have to decide which will make your debugging time more productive.

When working with video, do not process the entire video, tweak a constant, and re-process. Look at a single frame and update it while you adjust a slider. Then generate output for the whole sequence. This will cut your debugging time substantially.
4 Groups

The project groups that you chose are:

macintosh = 09jmc, 09dpf
rambo = 10ajs, 09wkj
reddelicious = 09msg,09twb,09ack_2
gala = 09hc, 10kl
cortland = 09sb,10msl
winston=09kaw_2,09cmz

You can access your files in SVN with:

svn co svn://graphics-svn.cs.williams.edu/371/5/fx-groupname