Creepy Face Tracking Portrait

Created by Tony DiCola

https://learn.adafruit.com/creepy-face-tracking-portrait

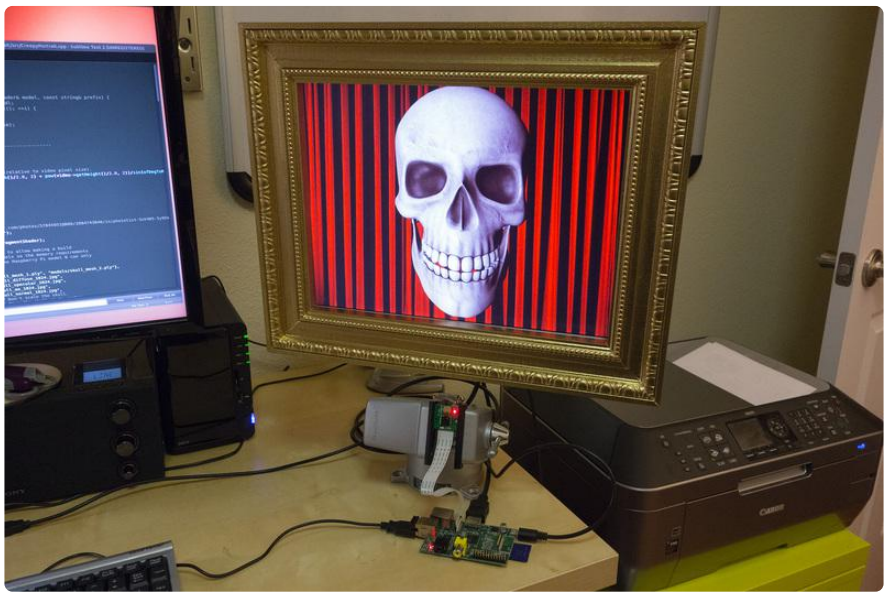
Last updated on 2023-08-29 02:24:30 PM EDT

Table of Contents

Overview

Have you ever seen a portrait where the face or eyes appear to follow your every movement? These are popular decorations at haunted houses that are typically based on the hollow face illusion (). In this illusion your brain is tricked into perceiving a concave face (i.e. hollow or caved-in) as actually standing out like normal face.

Inspired by creepy face following portraits, I decided to put a modern spin on the classic illusion by using real-time face detection. You can see from the video above and the photo below the project I created using a Raspberry Pi and camera running code based on openFrameworks () and OpenCV (). Follow this guide to learn how I built this creepy face tracking portrait!



Hardware

For this project you'll need the following hardware:

- Raspberry Pi (http://adafru.it/998) - I've tested this on and recommend the model B version with 512 MB of memory. The extra memory will help speed up compilation, and support loading more 3D model and texture data. Also, your Raspberry Pi will need to be connected to the internet to download the code and dependencies for this project.
- Raspberry Pi camera (http://adafru.it/1367) or USB webcam - The Pi camera has a fast refresh rate without much CPU usage, but a narrow field of view. A USB webcam typically has a wider field of view, but will consume more CPU and potentially run slowly. Stick with using a simple, low resolution webcam to help reduce the CPU usage--I had good results with this model from Frys ().

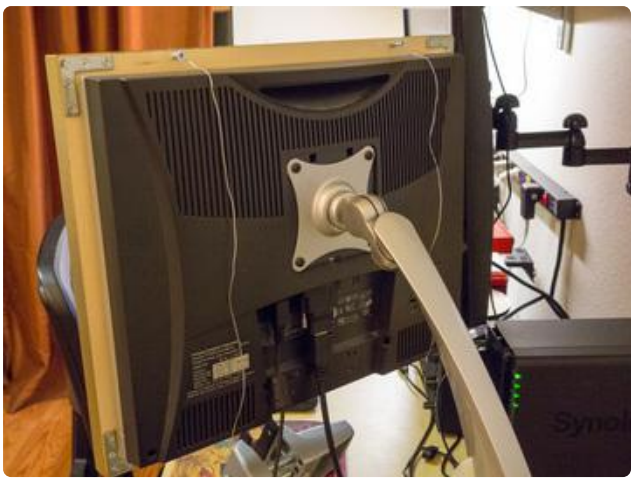
- Television or monitor - You need a display hooked up to the Raspberry Pi to show the creepy portrait. Any HDMI flat screen TV or computer monitor will work great!

As an alternative, the software for this project can run on your PC using a webcam. Right now the project only has a makefile for Linux-based systems, but the source code should be easily portable to a Windows or Mac development environment.

Consider building a custom frame for the monitor as a nice finishing touch. I followed this set of instructions () and had good results building a frame with wood molding and gold metallic spray paint. You can see some of the hardware and build steps to the left.

To build a frame you'll need these materials and tools:
Picture frame molding - Look in the crown & floor molding section of a big hardware store. Buy enough to fit the dimensions of your monitor, and remember you'll waste some on the edges to get the cuts to align. I used ~6 feet of material to build this 15" monitor frame.
Miter saw and guide box
Metal corner braces
Wood glue
Ratchet strap or other clamps
Sand paper (medium to fine grit)
Primer and paint
You can see to the left I hung the frame on the monitor by running baling wire between screws on opposite ends of the frame. This won't hold up to force or abuse, but will hold the frame nicely without making any permanent alterations to the monitor.

If you build a large enough frame you might even be able to drill a small hole in it to completely hide the camera, and hide the Raspberry Pi behind the monitor!



Continue on to install and learn about the software used in this project.

Software

If you want to download a pre-built binary version of the creepy portrait code follow the steps on this page.

If you're interested in compiling openFrameworks and the creepy portrait code (in particular if you want to do your own development with openFrameworks), skip this page and continue forward in the guide.

Binary Installation

First if you're new to using the Raspberry Pi, make sure you already have your Pi setup and running with a Raspbian operating system (Occidentalis () should work great too). You can follow these guides for more information on the initial Raspberry Pi setup:

- Lesson 1: Preparing an SD Card for your Raspberry Pi ()
- Lesson 2: Lesson 2. First Time Configuration ()

To download and install the application, open a terminal session (either through SSH or with a keyboard attached to the Raspberry Pi) and execute the following commands:

```
mkdir ~/creepyportrait
cd ~/creepyportrait
wget https://github.com/tdicola/creepyportrait/raw/master/
creepyportrait_1.0_raspberrypi.tar.gz
tar xvf creepyportrait_1.0_raspberrypi.tar.gz
sudo ./install_dependencies.sh
```

Answer yes to the questions about updating and installing packages that come up when the install_dependencies.sh script is run. This script is provided in openFrameworks and will install all the necessary libraries for running openFrameworks applications like the creepy portrait.

Next you'll need to ensure the Raspberry Pi GPU has at least 128 MB of memory available (if not, you won't see the 3D model textures render and instead get a black outline). You can change the split of memory in the raspi-config tool. Look at these instructions () for running the tool, and change the memory_split option to 128 MB for the GPU.

Usage

To run the program first make sure either the Raspberry Pi camera or a webcam is attached to your Raspberry Pi.

Note: If you're using the Raspberry Pi camera for the first time, make sure to follow these instructions () to enable it with the raspi-config command.

First run the program with no command line parameters by executing:

./creepyportrait

You should see the program usage and an error message about no video device being selected. If you're using a webcam, note the device ID number of the webcam you want to use.

To run the program with the Raspberry Pi camera execute:

./creepyportrait pi

Or to run the program with a webcam execute:

./creepyportrait (video device ID)

Where (video device ID) is the ID of the webcam from above. For example if you're using device ID 0 you would execute './creepyportrait 0' (without quotes).

It should take about 30-60 seconds for the program to start and display the skull. You should also see video from your camera in the upper left corner, and every ~2 seconds a green box appear over the largest detected face in the video. Make sure you have a decent amount of light on your face or else the detection won't be very reliable.

You can press the following buttons on the Raspberry Pi's keyboard to control the application:

- V - Hide or show the video in the upper left corner.
- M - If you're running with more than one 3D model (see further below), change between rendering different models. On the Raspberry Pi, for now only one model can be loaded in memory at a time.
- Escape or Ctrl-C - Close the application.

You can run the program with a different 3D model by specifying it in a second command line parameter. The possible values are:

- skull
- jackevil
- jackhappy
- all

Skull is the default model. Unfortunately even on the Raspberry Pi model B there isn't enough memory to load all models at once, so pick your favorite one and use it when you run on the Pi.

For example to run with the evil jack-o-lantern using the Raspberry Pi camera you would execute:

```
./creepyportrait pi jackevil
```

Continue on if you want to download and compile openFrameworks and the code for the creepy portrait. If you just want to run the creepy portrait you can stop here, you're done!

openFrameworks Install

This project is built on top of openFrameworks () which is an excellent toolkit for creative coding in the C++ programming language. The work of integrating together libraries like OpenGL, OpenCV (), and more is done for you, and the framework is ported to multiple platforms such as the PC, Mac, mobile devices, and recently the Raspberry Pi. To run the creepy portrait code you'll first need to follow these steps to install openFrameworks on your Raspberry Pi.

The openFrameworks website has a nice overview for running on the Raspberry Pi () that we'll use to install the framwork.

Note: Be aware the initial installation of openFrameworks on the Raspberry Pi will take about 1-2 hours of unattended time while the framework is compiled on the Pi. Set aside some time before you continue further.

To install openFrameworks on the Raspberry Pi, carefully follow the steps at the official installation instructions (). In particular, don't miss the step to configure the GPU/CPU memory split to 128 MB before compiling. I recommend installing openFrameworks to the /home/pi/openFrameworks folder like the installation guide suggests. Also, you can skip the optional steps at the end to setup a cross compiler or DISTCC for faster compilation.

Once openFrameworks is compiled, make sure you can compile one of the example applications by following the 'Compile your first app' step at the end of the instructions. Don't move on until you can successfully compile an openFrameworks application.

Continue on to compile and use the code for this project.

Code

Dependencies

You'll need to install at least version 4.7 of the GNU compiler collection (). The code in this project makes extensive use of the C++11 () language which requires at least GCC 4.7 to compile. At the time of writing this guide the Raspbian OS comes installed with only GCC 4.6. If you're not sure what version is installed, run the 'g++ --version' command in a terminal.

To install GCC 4.7 execute the following command in a terminal session on the Raspberry Pi:

```
sudo apt-get install gcc-4.7 g++-4.7
```

Note: If you're compiling this project to run on a Mac, unfortunately openFrameworks does not currently work with C++11 on Mac OSX. Support is in progress and you can follow this bug () for more details. This issue won't affect running on a Raspberry Pi or other platforms.

Next make sure you have the version control system git () installed. Git will be used to download the code for this project. To install git, run the following command:

```
sudo apt-get install git
```

Compilation

Once the dependencies are met you can download and compile the code for this project from its home on github (). In a terminal session navigate to the apps/myApps directory beneath the openFrameworks root folder. If you installed openFrameworks to the default location in the previous install steps, this will be your /home/pi/openFrameworks/apps/myApps folder.

Inside the myApps folder, execute the following command to download the project code:

```
git clone https://github.com/tdicola/creepyportrait ()
```

Navigate into the creepy portrait subfolder and compile the project by executing these commands:

```
cd creepyportrait
make
```

The project should compile successfully after about 5-10 minutes. If you run into an error, check that you've followed the steps to successfully compile openFrameworks and a sample application.

Usage

Skip back to the Software () page in this guide to learn about the usage of the program.

Continue on to learn about how to modify the program and use it for future projects.

Future Work

This project demonstrates near real-time face detection from a video camera on the Raspberry Pi. You can use the code in this project as a guide for integrating face detection into your own Raspberry Pi-based projects. To help with this, here's a quick explanation of some of the important files in the code:

- CreepyPortrait.h - This file defines the internal state used by the application. You can modify the public members to change some of the functionality. Look for the explanation of each attribute in the comments. Also note main.cpp has two configuration functions, one for running on the Raspberry Pi and another for running on the PC--look at how each function sets the internal state like the delay in face detection, face detection buffer size, etc.
- VideoSource.h & VideoSource.cpp - These files define a simple facade to wrap the openFrameworks webcam and Raspberry Pi camera video grabbers into a similar interface. If you're using the Raspberry Pi camera, be aware the addon to integrate it with openFrameworks is in an early beta stage and not yet part of the openFrameworks project. You can find the latest code for the Raspberry Pi camera addon at its github homepage (). Special thanks to Jason Van Cleave for publishing this camera code!
- VideoFaceDetector.h & VideoFaceDetector.cpp - These files define a class that uses the OpenCV addon to detect faces from frames in a video with a Haar feature-based cascade classifier (). If you're running the face detection at a high rate, like more than once a second, you might notice noise from the center of the detected face slightly moving around. To smooth out this noise this class implements a buffer so previous frames are used to calculate an average detected face location. If you're running on a Raspberry Pi which can't detect faces as quickly this buffering isn't as necessary.

Here are a few more tips I learned if you plan to do face detection on the Raspberry Pi:

- The size of video you're capturing will directly impact the performance of the face detection. The smaller the video the faster the face detection will run. I found 160x120 resolution video is a good trade-off of speed and quality.

- Build your app to work with ~500-800 milliseconds of latency when detecting faces. To appear smooth my code runs the face detection every 2 seconds and then animates moving the model in the time it's waiting for the next face detection.
- Watch out for memory consumption, especially if you're doing 3D rendering. You can quickly consume all the memory on the Pi with large models or textures. Scale assets down to as small a size as possible.

Have fun integrating face detection into your own Raspberry Pi project!