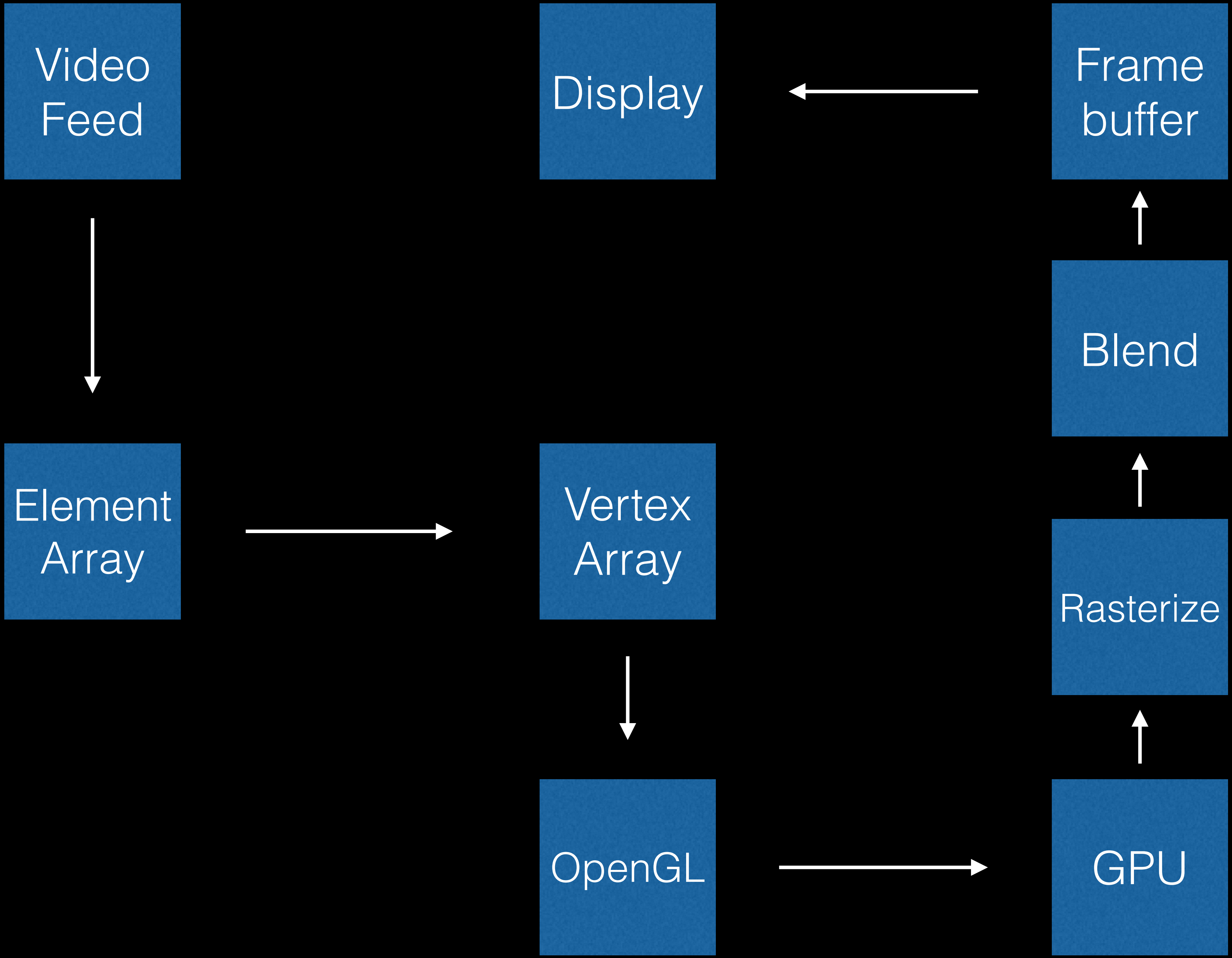
Real-Time Video Processing with OpenGL

Leo Schultz

Pipeline


Video Feed
Display
Frame buffer
Blend
Element Array
Vertex Array
Rasterize
OpenGL
GPU

Sequence

Get Device with Video Grabber

Pass Device Pixels to Array

Manipulate the Pixel Array

Pass Array to GPU for computation

Render Computation to Display

Frameworks

- OpenFramework
    - Cocoa
    - GLUT
    - OpenGL
    - CoreVideo

Open Framework

```
// Callable Methods from framework

ofSetVerticalSync(true);

<ofVideoDevice>

// ofSomeFunction();
```

Declaration

```cpp
#pragma once

#include "ofMain.h"

enum videoMode{
    STANDARD_MODE = 0
    INVERTED_MODE,
    OUTLINE_MODE,
    TOON_MODE
};

class ofApp : public ofBaseApp

    public:
      void setup();
      void update();
      void draw();

      void keyPressed(int key);
      void keyReleased(int key);
      void mouseMoved(int x, int y );
      void mouseDragged(int x, int y, int button);
      void mousePressed(int x, int y, int button);
      void mouseReleased(int x, int y, int button);
      void windowResized(int w, int h);
      void dragEvent(ofDragInfo dragInfo);
      void gotMessage(ofMessage msg);

        void setMode(videoMode newMode);

        ofVideoGrabber              vidGrabber;
        unsigned char *      videoInverted;
        ofTexture            videoTexture;
        int                         camWidth;
        int                         camHeight;
        int                 frameNum;
        string              currentModeStr;
        videoMode           currentMode;
};
```

Setup

```
void ofApp::setup(){
    camWidth     = 2880;   // try to grab at this size.
    camHeight        = 1800;

    //we can now get back a list of devices.
    vector<ofVideoDevice> devices = vidGrabber.listDevices();

    for int i = 0; i < devices.size(); i++){
        cout << devices[i].id << ": " << devices[i].deviceName;
        if( devices[i].bAvailable ){
            cout << endl;
        }else{
            cout << " - unavailable " << endl;
        }
    }

    vidGrabber.setDeviceID(0);
    vidGrabber setDesiredFrameRate(60);
    vidGrabber.initGrabber(camWidth,camHeight);

    videoInverted    = new unsigned char camWidth camHeight*3];
    videoTexture allocate camWidth camHeight, GL_RGB);
    ofSetVerticalSync(true);
}
```

Filters

```
vidGrabber.update();

int totalPixels = camWidth*camHeight*3;
unsigned char * pixels = vidGrabber.getPixels();

if( currentMode == INVERTED_MODE ){
    for (int i = 0; i < totalPixels; i++){
        frameNum ++;
        videoInverted[i] = (int ofRandom(150 210) - pixels[i];
    }
    videoTexture.loadData(videoInverted, camWidth,camHeight, GL_RGB);
}
else if( currentMode == OUTLINE_MODE ){
    for  int i = 0; i < totalPixels; i++){
        frameNum ++;
        videoInverted[i] = 155 + pixels[i];
    }
    videoTexture.loadData(videoInverted, camWidth camHeight, GL_RGB);
}
else if( currentMode == TOON_MODE  ){
    for  int i = 0; i < totalPixels; i++){
        frameNum ++;
        videoInverted[i] = 20 * pixels[i];
    }
    videoTexture.loadData(videoInverted, camWidth,camHeight, GL_RGB);
}
```

Performance

Standard Video Output at **60 fps**

iMac - With filter after Parallelization = **22 fps**

Macbook - With filter after Parallelization = 15fps

iMac - With filter before Parallelization = **7 fps**

Macbook - With filter before Parallelization = 3 fps