

MORE THAN TECHNICAL

On software, code, the internet and more.

FEBRUARY 17, 2015 BY ROY

Augmented Reality on libQGLViewer and OpenCV-OpenGL tips [w/code]

You already know [I love libQGLViewer](#). So here a snippet on how to do AR in a QGLViewer widget. It only requires a couple of tweaks/overloads to the plain vanilla widget setup (using the matrices properly, disable the mouse binding) and it works.

The major problems I recognize with getting a working AR from OpenCV's intrinsic and extrinsic camera parameters are their translation to OpenGL. I saw a whole lot of solutions online, and I contributed [from my own experience a while back](#), so I want to reiterate here again in the context of libQGLViewer, with a couple extra tips.

Intrinsic parameters and the projection matrix

We all know the intrinsic parameter matrix that is obtained from a calibration process:

$$\begin{pmatrix} \alpha & 0 & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

Well it could be approximated with mock values if you know the frame size, but you cannot calibrate the camera.

For example for a 640x480 frame the matrix would be:

$$\begin{pmatrix} 640 & 0 & 320 \\ 0 & 640 & 240 \\ 0 & 0 & 1 \end{pmatrix}$$

Using $\alpha = \beta = \max(\text{width}, \text{height})$ as the focal length and pixel size dependent parameter (this number is not the focal length!).

If you want precision, calibrate the camera or get the calibration matrix from somewhere, but if you just want to hack - this is a rough approximation.

Getting the projection matrix that is derived from this matrix is fairly simple. It's the following 4x4 matrix:

$$\begin{pmatrix} \frac{f_x}{c_x} & 0 & 0 & 0 \\ 0 & \frac{f_y}{c_y} & 0 & 0 \\ 0 & 0 & \frac{-(far+near)}{far-near} & \frac{-2.0*far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

And in code:

```

1  Mat_<double> persp(4,4); persp.setTo(0);
2
3  // http://kgeorge.github.io/2014/03/08/calculating-opengl-perspective/
4  double fx = camMat.at<float>(0,0);
5  double fy = camMat.at<float>(1,1);
6  double cx = camMat.at<float>(0,2);
7  double cy = camMat.at<float>(1,2);
8  persp(0,0) = fx/cx;
9  persp(1,1) = fy/cy;
10 persp(2,2) = -(far+near)/(far-near);
11 persp(2,3) = -2.0*far*near / (far-near);
12 persp(3,2) = -1.0;
13
14 cout << "perspective m \n" << persp << endl;
15
16 persp = persp.t(); //to col-major for OpenGL
17 glMatrixMode(GL_PROJECTION);
18 glLoadMatrixd((double*)persp.data);

```

It works, now let's keep going.

Extrinsic parameters

Another point I see people have a trouble getting through is taking the output of `solvePnP()` and getting the modelview matrix for OpenGL.

Many of the guides say "simply use R and t as they are", but that's not exactly the case... we need to flip the Y and Z axis because of OpenCV and OpenGL conventions.

```

1  cv::Mat Rvec,Tvec;
2  cv::solvePnP(ObjPoints, Points(trackedFeatures), camMat, Mat(), rai
3  raux.convertTo(Rvec,CV_32F);
4  taux.convertTo(Tvec ,CV_64F);
5
6  Mat Rot(3,3,CV_32FC1);
7  Rodrigues(Rvec, Rot);
8
9  // [R | t] matrix
10 Mat_<double> para = Mat_<double>::eye(4,4);
11 Rot.convertTo(para(Rect(0,0,3,3)),CV_64F);
12 Tvec.copyTo(para(Rect(3,0,1,3)));
13
14 Mat cvToGl = Mat::zeros(4, 4, CV_64F);
15 cvToGl.at<double>(0, 0) = 1.0f;
16 cvToGl.at<double>(1, 1) = -1.0f; // Invert the y axis
17 cvToGl.at<double>(2, 2) = -1.0f; // invert the z axis
18 cvToGl.at<double>(3, 3) = 1.0f;
19
20 para = cvToGl * para;
21
22 Mat(para.t()).copyTo(modelview_matrix); // transpose to col-major t

```

This should get you going.

Remember raux and taux could be used for the processing of the next frame as an initial guess.

Setting up QGLViewer

First step is to get the projection matrix uploaded. This has to be done via the GLViewer's camera() object. Their documentation suggest we subclass it, so here's how it's done:

```

1  class OpenCVCamera : public qglviewer::Camera {
2  public:
3      Mat camMat;
4
5      virtual void loadProjectionMatrix(bool reset) const {
6          static Mat_<double> persp;
7          double near = 1, far = 100.0;
8
9          glMatrixMode(GL_PROJECTION);
10         if(persp.empty()) {
11             persp.create(4,4); persp.setTo(0);
12
13             // http://kgeorge.github.io/2014/03/08/calculating-oper
14             double fx = camMat.at<float>(0,0);

```

```

15         double fy = camMat.at<float>(1,1);
16         double cx = camMat.at<float>(0,2);
17         double cy = camMat.at<float>(1,2);
18         persp(0,0) = fx/cx;
19         persp(1,1) = fy/cy;
20         persp(2,2) = -(far+near)/(far-near);
21         persp(2,3) = -2.0*far*near / (far-near);
22         persp(3,2) = -1.0;
23
24         cout << "perspective m \n" << persp << endl;
25
26         persp = persp.t(); //to col-major
27     }
28     glLoadMatrixd((double*)persp.data);
29 }
30 };

```

Apparently the loadProjectionMatrix() functions gets called every frame, so I optimized by caching the "persp" matrix and thereafter simply use the prepared matrix.

This needs to be then initialized in thw QGLWidget's init():

```

1  class MyQGLViewer : public QGLViewer {
2  // ...
3  private:
4      QBasicTimer*      frameTimer;
5      RS::OpenCVGLTexture ocv_tex;
6      Mat               frame;
7      Mat               camMat;
8  // ...
9  public:
10 // ...
11     virtual void init() {
12         // Enable GL textures
13         glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
14         glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
15         // Nice texture coordinate interpolation
16         glHint( GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST );
17
18         ocv_tex = RS::MakeOpenCVGLTexture(frame);
19
20         setFixedHeight(frame.rows);
21         setFixedWidth(frame.cols);
22
23         clearMouseBindings();
24
25         frameTimer->start(1,this);
26
27         OpenCVCamera* c = new OpenCVCamera;
28         c->camMat = camMat;
29         setCamera(c);
30     }
31 // ...
32 };

```

Now I got a few more things going there besides the camera(), first there's the QBasicTimer.

This timer fires every 1ms (in reality this should be set to 30ms) and will upload the frame to the GPU memory to be shown as a texture, we'll see that in a moment.

Then there's the OpenCV-OpenGL texture object that's my own implementation, to make life easier when using OpenCV Mats and OpenGL textures. You can get the gist here: <https://gist.github.com/royshil/5b96b6a1797e12fcef8d>

One more thing, I set the widget size to be fixed width and height as well as remove the mouse bindings. This being an AR program, the mouse should have control of the camera and the window size should be set to avoid having to create the projection matrix again.

Drawing is trivial, and partially based on the QGLViewer background image example: <http://www.libqglviewer.com/examples/contribs.html#backgroundImage>

Here is the complete code in a gist:

```

1  /*
2   * SimpleARQGLViewer.cpp
3   * Creating an AR application with QGLViewer
4   *
5   * Created by Roy Shilkrot on 2/16/2015
6   * Copyright 2015 Roy Shilkrot. All rights reserved.
7   *
8   * Permission is hereby granted, free of charge, to any person obtaining a copy
9   * of this software and associated documentation files (the "Software"), to deal
10  * in the Software without restriction, including without limitation the rights
11  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
12  * copies of the Software, and to permit persons to whom the Software is
13  * furnished to do so, subject to the following conditions:
14  *
15  * The above copyright notice and this permission notice shall be included in
16  * all copies or substantial portions of the Software.
17  *
18  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
19  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
20  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
21  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
22  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
23  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
24  * THE SOFTWARE.
25  *
26  */

```

```
27
28  #include <opencv2/opencv.hpp>
29
30  #include <iostream>
31  #include <algorithm>
32
33  #include "OGL_OCV_common.h"
34  #include <GLUT/glut.h>
35
36  #include <qapplication.h>
37  #include <QBasicTimer>
38  #include <QGLViewer/qglviewer.h>
39
40  using namespace cv;
41  using namespace std;
42
43  class Tracker {
44  private:
45      //...
46      Mat raux,taux; //keep R and t for next frame
47      //...
48  public:
49      //...
50
51      void process(const Mat& frame) {
52          // track 2D features and match them to known 3D points
53      }
54
55      //...
56
57      void calcModelViewMatrix(Mat& modelview_matrix, const Mat& camMat) {
58          vector<Point2f> ObjPoints,ImagePoints;
59          // Setup ObjPoints and ImagePoints so they correspond 3D -> 2D
60
61          cv::Mat Rvec,Tvec;
62          cv::solvePnP(ObjPoints, ImagePoints, camMat, Mat(), raux, taux, !raux.empty());
63          raux.convertTo(Rvec,CV_32F);
64          taux.convertTo(Tvec ,CV_64F);
65
66          Mat Rot(3,3,CV_32FC1);
67          Rodrigues(Rvec, Rot);
68
69          // [R | t] matrix
70          Mat_<double> para = Mat_<double>::eye(4,4);
71          Rot.convertTo(para(Rect(0,0,3,3)),CV_64F);
```

```

72     Tvec.copyTo(para(Rect(3,0,1,3)));
73
74     Mat cvToGl = Mat::zeros(4, 4, CV_64F);
75     cvToGl.at<double>(0, 0) = 1.0f;
76     cvToGl.at<double>(1, 1) = -1.0f; // Invert the y axis
77     cvToGl.at<double>(2, 2) = -1.0f; // invert the z axis
78     cvToGl.at<double>(3, 3) = 1.0f;
79
80     para = cvToGl * para;
81
82     Mat(para.t()).copyTo(modelview_matrix); // transpose to col-major for OpenGL
83 }
84 };
85
86 class OpenCVCamera : public qglviewer::Camera {
87 public:
88     Mat camMat;
89
90     virtual void loadProjectionMatrix(bool reset) const {
91         static Mat_<double> persp;
92         double near = 1, far = 100.0;
93
94         glMatrixMode(GL_PROJECTION);
95         if(persp.empty()) {
96             persp.create(4,4); persp.setTo(0);
97
98             // http://kgeorge.github.io/2014/03/08/calculating-opengl-perspective-matrix-f
99             double fx = camMat.at<float>(0,0);
100             double fy = camMat.at<float>(1,1);
101             double cx = camMat.at<float>(0,2);
102             double cy = camMat.at<float>(1,2);
103             persp(0,0) = fx/cx;
104             persp(1,1) = fy/cy;
105             persp(2,2) = -(far+near)/(far-near);
106             persp(2,3) = -2.0*far*near / (far-near);
107             persp(3,2) = -1.0;
108
109             cout << "perspective m \n" << persp << endl;
110
111             persp = persp.t(); //to col-major
112         }
113         glLoadMatrixd((double*)persp.data);
114     }
115 };
116

```

```
117 class Viewer : public QGLViewer
118 {
119 protected :
120     VideoCapture      vc;
121     Mat               frame;
122     Mat               orig_gray;
123     RS::OpenCVGLTexture ocv_tex;
124     Tracker           tracker;
125     Mat_<float>        camMat;
126     Mat_<double>       modelViewMatrix;
127     QBasicTimer*      frameTimer;
128
129 public:
130     Viewer() {
131
132         vc.open("myvideo.MOV");
133         if(!vc.isOpened()) {
134             cerr << "can't open video\n";
135         } else {
136             Mat frame_,orig,orig_warped,tmp;
137             vc >> frame_;
138             if(frame_.empty()) {
139                 cerr << "can't get first frame\n";
140             } else {
141                 frame_.copyTo(frame);
142                 float f = std::max(frame.cols,frame.rows);
143                 camMat = (Mat_<float>(3,3) <<  f,      0,      frame.cols/2,
144                                     0,      f,      frame.rows/2,
145                                     0,      0,      1);
146
147                 frameTimer = new QBasicTimer();
148             }
149         }
150     }
151     ~Viewer() {
152         frameTimer->stop();
153         delete frameTimer;
154     }
155
156     virtual void draw() {
157         drawBackground();
158
159         glLoadMatrixd((double*)modelViewMatrix.data);
160
161         glPushMatrix();
```



```
162     glDisable(GL_LIGHTING);
163     glColor3f(1,0,0);
164     glTranslatef(0,0,-0.3);
165     glutWireCube( 0.6 );
166     glEnable(GL_LIGHTING);
167     glPopMatrix();
168 }
169
170 virtual void init() {
171     // Enable GL textures
172     glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
173     glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
174     // Nice texture coordinate interpolation
175     glHint( GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST );
176
177     ocv_tex = RS::MakeOpenCVGLTexture(frame);
178
179     setFixedHeight(frame.rows);
180     setFixedWidth(frame.cols);
181
182     clearMouseBindings();
183
184     frameTimer->start(1,this);
185     detectorTimer->start(1,this);
186
187     OpenCVCamera* c = new OpenCVCamera;
188     c->camMat = camMat;
189     setCamera(c);
190 }
191
192 void timerEvent(QTimerEvent *timer) {
193     Mat frame_;
194     vc >> frame_;
195     if(frame_.empty()) return;
196
197     frame_.copyTo(frame);
198
199     tracker.process(frame);
200     tracker.calcModelViewMatrix(modelViewMatrix,camMat);
201
202     ocv_tex.set(frame);
203
204     updateGL();
205 }
206
```

```
207 void drawBackground()
208 {
209
210     glDisable(GL_LIGHTING);
211     glEnable(GL_TEXTURE_2D);
212     glColor3f(1,1,1);
213
214     startScreenCoordinatesSystem(true);
215
216     // Draws the background quad
217     RS::drawOpenCVImageInGLOnlyQuad(ocv_tex,width(),height());
218
219     stopScreenCoordinatesSystem();
220
221     // Depth clear is not absolutely needed. An other option would have been to draw the
222     // QUAD with a 0.999 z value (z ranges in [0, 1[ with startScreenCoordinatesSystem()).
223     glClear(GL_DEPTH_BUFFER_BIT);
224     glDisable(GL_TEXTURE_2D);
225     glEnable(GL_LIGHTING);
226 }
227
228 };
229
230 int main(int argc, char** argv) {
231
232     // Read command lines arguments.
233     QApplication application(argc,argv);
234
235     cout << "running...\n";
236
237     // Instantiate the viewer.
238     Viewer viewer;
239
240     viewer.setWindowTitle("Simple AR QGLViewer");
241
242     // Make the viewer window visible on screen.
243     viewer.show();
244
245     // Run main loop.
246     return application.exec();
247
248 }
```

[SimpleARQGLViewer.cpp](#) hosted with ❤ by [GitHub](#)

[view raw](#)

And here is a screen shot.. (I'm using my own natural features marker tracker)



Enjoy!

Roy



3D, AUGMENTED REALITY, CODE, GRAPHICS, OPENCV, OPENGL, PROGRAMMING, QT, TRACKING, VIDEO, VISION

AUGMENTED REALITY

0 Comments

More Than Technical

 Login ▾ Recommend Tweet Share

Sort by Best ▾



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 


Name

Be the first to comment.


ALSO ON MORE THAN TECHNICAL

Cross-compile latest Tensorflow (1.5+) for the ...


1 comment • a year ago

 **Karthik** — Excellent information, greatly consolidated. Thanks for the post :) I have a query, can ...**Revisiting graph-cut segmentation with SLIC and ...**


2 comments • a year ago

 **Megan** — In your 2010 work the final result is the extracted image (attached), not just a ...**Android Camera2 Touch-to-Focus**

4 comments • 2 years ago

 **Swastik Devs** — if i keep the repeat off then the preview gets stuck and if i keep it continuous ...**Finding FFMPEG with CMake**

1 comment • 4 years ago

 **ahmet urun** — dudeeee this is awesooooome thanks broo . Subscribe  Add Disqus to your site Add Disqus Add Disqus' Privacy Policy Privacy Policy Privacy