UNIVERSITÉ DE LORRAINE | IUT Saint-Dié-des-Vosges
Services et Réseaux de Communication

Année 2015-2016

Projet tuteuré 4

Implémentation d'une scène avec réalité augmentée sur une photo

Tuteur : Pierre-Fréderic Villard

CHAMARD Esteban INFO 2

Sommaire

Introduction

Problématique :

« Implémentation d'une scène avec réalité augmentée sur une photo »

Avec les contraintes suivantes :

- Utilisation de la bibliothèque PoLAR (polar.inria.fr)
- Utilisation basique de C++

Qu'est-ce que la réalité augmentée ?

La réalité augmentée vise à ajouter des éléments virtuels au monde qui nous entoure, en offrant à l'utilisateur la possibilité d'être immergé dans cet environnement mixte. (Source :loria.fr)

Qu'est-ce que la bibliothèque PoLAR ?

PoLAR (Portable Library for Augmented Reality) est un framework qui a pour but d'aider la création d'application de réalité augmenté, la visualisation d'image et l'imagerie médicale. PoLAR est fait pour offrir des outils utiles sans avoir le besoin d'être un spécialiste. Il est écrit en C++ et est publié sous la Licence GNU GPL.

Quel est l'objectif du projet ?

Remplir un ensemble de tâche :

- T1 : Compiler la bibliothèque PoLAR
- T2 : Charger une photo
- T3 : Mettre un (ou plusieurs) objet 3D sur la photo avec la bonne matrice de protection
- T4 : Mettre un (ou plusieurs) objet fantôme pour avoir de l'ombrage
- T5 : Mettre un (ou plusieurs) objet fantôme pour avoir de l'occlusion
- T6 : Ajouter des comportements dynamiques
- T7 : Ajouter des comportements évènementiels
- T8 : Faire un gif animé montrant les différentes étapes de création de la scène

Installation de PoLAR

Système : Ubuntu 15.10 64 Bits

Les packages suivant sont nécessaires :

Qt5 packages:

```
esteban@esteban-linux:~$ sudo apt-get install qt5-default qttools5-dev-tools lib
qt5opengl5-dev qtmultimedia5-dev libqt5multimedia5-plugins qtdeclarative5-dev li
bqt5svg5-dev
```

OpenSceneGraph packages:

```
esteban@esteban-linux:~$ sudo apt-get install openscenegraph libopenscenegraph-dev
```

Gstreamer packages:

```
esteban@esteban-linux:~$ sudo apt-get libgstreamer0.10-0 libgstreamer0.10-dev gstreamer-tools
```

Doxygen documentation :

```
esteban@esteban-linux:~$ sudo apt-get doxygen graphviz
```

Calcul de la matrice de projection

Pour calculer la matrice de projection, il a été décidé utiliser un script python, après un cuisant échec sur Java (problème d'inversion de la matrice insoluble, ou du moins plus chronophage que de redévelopper la solution sous python).

Ensuite il faut transcrire les différentes opérations de matrice données dans le sujet, en précisant les 4 extrémités de la surface à laquelle on doit appliquer la matrice (ici trouvé via gimp, n'ayant pas réussi à utiliser l'outil préconisé dans le sujet). Ainsi que les dimensions de l'image (en pixel) de fond et la largeur de notre surface réelle (en cm).
Les opérations à appliquer :

On calcule la position des points centrés sur l'image

$$Q1' = \begin{bmatrix} Q1_x - w/2 \\ Q1_y - h2 \\ 1 \end{bmatrix}; Q2' = \begin{bmatrix} Q2_x - w/2 \\ Q2_y - h2 \\ 1 \end{bmatrix}; Q3' = \begin{bmatrix} Q3_x - w/2 \\ Q3_y - h2 \\ 1 \end{bmatrix}; Q4' = \begin{bmatrix} Q4_x - w/2 \\ Q4_y - h2 \\ 1 \end{bmatrix}$$

On calcule des lignes de fuites

$$l1 = \begin{bmatrix} Q1'_x \\ Q1'_y \\ 1 \end{bmatrix} \otimes \begin{bmatrix} Q2'_x \\ Q2'_y \\ 1 \end{bmatrix}; l2 = \begin{bmatrix} Q4'_x \\ Q4'_y \\ 1 \end{bmatrix} \otimes \begin{bmatrix} Q3'_x \\ Q3'_y \\ 1 \end{bmatrix}; l3 = \begin{bmatrix} Q2'_x \\ Q2'_y \\ 1 \end{bmatrix} \otimes \begin{bmatrix} Q3'_x \\ Q3'_y \\ 1 \end{bmatrix}; l4 = \begin{bmatrix} Q1'_x \\ Q1'_y \\ 1 \end{bmatrix} \otimes \begin{bmatrix} Q4'_x \\ Q4'_y \\ 1 \end{bmatrix}$$

On calcule les intersections V et W

$$V^H = l1 \otimes l2; W^H = l3 \otimes l4; V = \begin{bmatrix} V_x^H/V_z^H \\ V_y^H/V_z^H \\ 1 \end{bmatrix}; W = \begin{bmatrix} W_x^H/W_z^H \\ W_y^H/W_z^H \\ 1 \end{bmatrix}$$

On calcule la distance focale

$$f = \sqrt{-(V_x.W_x + V_y.W_y)}$$

A partir de laquelle on calcule la matrice intrinsèque

$$K = \begin{bmatrix} f & 0 & w/2 \\ 0 & f & h/2 \\ 0 & 0 & 1 \end{bmatrix}$$

On obtient la matrice d'homographie par la multiplication de la transposé de A et de Qi' tel que :

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & -Q2'_x & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & -Q2'_y & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & -Q3'_x & -Q3'_x \\ 0 & 0 & 0 & 1 & 1 & 1 & -Q3'_y & -Q3'_y \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & -Q4'_x \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & -Q4'_y \end{bmatrix} ; Qi' = \begin{bmatrix} Q1'_x \\ Q1'_y \\ Q2'_x \\ Q2'_y \\ Q3'_x \\ Q3'_y \\ Q4'_x \\ Q4'_y \end{bmatrix}$$

Puis on calcule B :

$$B = \begin{bmatrix} 1/f & 0 & 0 \\ 0 & 1/f & 0 \\ 0 & 0 & 1 \end{bmatrix} .h = \begin{bmatrix} \vdots & \vdots & \vdots \\ B1 & B2 & B3 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

Et enfin on obtient la matrice de projection en multipliant la matrice intrinsèque à E tel que :

$$E = \begin{bmatrix} \vdots & \vdots & \vdots & \\ R1 & R2 & R3 & T \\ \vdots & \vdots & \vdots & \end{bmatrix}$$

avec les valeurs suivantes :

$$\begin{array}{lcl} R1 & = & B1/\|B1\| \\ R3 & = & R1 \otimes B2/(\|R1 \otimes B2\|) \\ R2 & = & R3 \otimes R1 \\ \lambda & = & \|B1\|/s \\ t_{calcule} & = & \|B2\|/\lambda \\ T & = & B3/\lambda \end{array}$$

Création d'un nouveau projet utilisant PoLAR

Le projet n'étant pas dans le dossier de PoLAR, il a fallu définir le fichier de source des librairies dans le CMakeLists.txt en ajoutant les lignes :

set(PoLAR_LIBRARY "/home/esteban/polar/lib/libPoLAR.so")

set(PoLAR_SOURCE_DIR "/home/esteban/polar")

- remplacer 'runPolar' par le nom de votre projet
- enlever les appels à OpenCV:

```
find_package(OpenCV REQUIRED core imgproc highgui calib3d xfe

set(OPEN_CV_LIBS
  ${OpenCV_LIBS}
  #opencv_core
  #opencv_features2d
  #opencv_highgui
  #opencv_calib3d
  #opencv_xfeatures2d
  #opencv_imgproc
  #opencv_imgcodecs
  #stdc++

include_directories(${OpenCV_INCLUDE_DIRS})}
```

- lister vos propres fichiers sources à la place de ceux de runPoLAR :

```
set(SRCS
  LoadImageDialog.cpp
  LoadImageDialog.h
  PoseViewer.cpp
  PoseViewer.h
  Tracker.cpp
  Tracker.h
  VideoViewer.cpp
  VideoViewer.h
  Interface.cpp
  Interface.h
)
```

Annexes

Annexe A : Code pour le calcul de la matrice

```
#!/usr/bin/python
#coding: utf-8
import math
import numpy

#Valeurs des points sur l'image
q1=[276,265]
q2=[556,265]
q3=[586,365]
q4=[228,356]

h = 563.0 #hauteur en pixel de l'image
w = 750.0  #largeur en pixel de l'image

s = 50.0 #largeur de la table en Dm

q1p = [q1[0]-w/2.0, q1[1]-h/2.0, 1]
q2p = [q2[0]-w/2.0, q2[1]-h/2.0, 1]
q3p = [q3[0]-w/2.0, q3[1]-h/2.0, 1]
q4p = [q4[0]-w/2.0, q4[1]-h/2.0, 1]

def prod_vectoriel(u, v):
    x = u[1]*v[2] - u[2]*v[1]
    y = u[2]*v[0] - u[0]*v[2]
    z = u[0]*v[1] - u[1]*v[0]
    return [x, y, z]

def norme(u):
    return math.sqrt(u[0]*u[0]+u[1]*u[1]+u[2]*u[2])

def div(u, x):
    return [u[0]/x, u[1]/x, u[2]/x]

def print_matrix(m):
    aff = ""
    for ligne in m:
        for nb in ligne:
            aff+= str(nb) + " "
        aff+= "\n"
    print aff


l1 = prod_vectoriel([q1p[0],q1p[1],1], [q2p[0],q2p[1],1])
l2 = prod_vectoriel([q4p[0],q4p[1],1], [q3p[0],q3p[1],1])
l3 = prod_vectoriel([q2p[0],q2p[1],1], [q3p[0],q3p[1],1])
l4 = prod_vectoriel([q1p[0],q1p[1],1], [q4p[0],q4p[1],1])

vh = prod_vectoriel(l1, l2)
wh = prod_vectoriel(l3, l4)
v = [vh[0]/vh[2], vh[1]/vh[2], 1]
W = [wh[0]/wh[2], wh[1]/wh[2], 1]

if (v[0]*W[0]+v[1]*W[1])>0:
```

```
    print "impossibilité physique"

f = math.sqrt(-(v[0]*W[0]+v[1]*W[1]))

k = [[f, 0, w/2.0],
     [0, f, h/2.0],
     [0,0,1]]

qip = [[q1p[0]],
       [q1p[1]],
       [q2p[0]],
       [q2p[1]],
       [q3p[0]],
       [q3p[1]],
       [q4p[0]],
       [q4p[1]]]

A = [[0,0,1,0,0,0,0,0],
     [0,0,0,0,0,1,0,0],
     [1,0,1,0,0,0,-q2p[0],0],
     [0,0,0,1,0,1,-q2p[1],0],
     [1,1,1,0,0,0,-q3p[0],-q3p[0]],
     [0,0,0,1,1,1,-q3p[1],-q3p[1]],
     [0,1,1,0,0,0,0,-q4p[0]],
     [0,0,0,0,1,1,0,-q4p[1]]]

A1 = numpy.linalg.inv(A)


temp = numpy.dot(A1, qip)
h=[[temp[0][0],temp[1][0],temp[2][0]],
   [temp[3][0],temp[4][0],temp[5][0]],
   [temp[6][0],temp[7][0],1]]


tmp = [[1/f,0,0],[0,1/f,0],[0,0,1]]
B = numpy.dot(tmp, h)
B1 = [B[0][0], B[1][0], B[2][0]]
B2 = [B[0][1], B[1][1], B[2][1]]
B3 = [B[0][2], B[1][2], B[2][2]]

R1 = div(B1, norme(B1))
R3 = div(prod_vectoriel(R1, B2), norme(prod_vectoriel(R1, B2)))
R2 = prod_vectoriel(R3, R1)
l = norme(B1)/s
t_calc = norme(B2)/l
T = div(B3, l)
E = [[R1[0],R2[0],R3[0],T[0]],
     [R1[1],R2[1],R3[1],T[1]],
     [R1[2],R2[2],R3[2],T[2]]]

M = numpy.dot(k,E)
print_matrix(M)
```

Annexe B : Code de la scène

```
#include <QtWidgets/QApplication> // for Qt functions
#include "Viewer.h" // PoLAR::Viewer header
#include "Image.h"
#include "Util.h"

#include <iostream>
#include <QApplication>
#include <osg/Geode>
#include <osg/Geometry>
#include <osgDB/ReadFile>
#include <osgUtil/SmoothingVisitor>
#include <osgUtil/Optimizer>
#include <osg/io_utils>

int main(int argc, char** argv)
{
    // Create the Qt application
    QApplication app(argc, argv);
    PoLAR::Viewer viewer(0, "Viewer", 0, true);
    osg::ref_ptr<PoLAR::Image<unsigned char> > myImage;

    //Chargement de l'image
    myImage = new PoLAR::Image<unsigned char>("../salon.jpg", true, 1);
    viewer.setBgImage(myImage);
    viewer.bgImageOn();

    //Redimmensioner
    int width = myImage->getWidth();
    int height = myImage->getHeight();
    viewer.resize(width, height);
    osgUtil::SmoothingVisitor smooth;
    osgUtil::Optimizer optimizer;


    //chargement du presse agrume
    osg::ref_ptr<osg::Node> loadedModel2;
    loadedModel2 = osgDB::readNodeFile("../squeezer.obj");

    optimizer.optimize(loadedModel2.get());
    loadedModel2->accept(smooth);

    //Création de l'objet
    osg::ref_ptr<PoLAR::Object3D> obj2 = new PoLAR::Object3D(loadedModel2.get(), true,
true);

    //Modification de l'objet
    obj2->scale(2.0f);
    obj2->translate(10,10,-2);
    obj2->rotate(M_PI, 0, 1, 0);


    //chargement du balai
    osg::ref_ptr<osg::Node> loadedModel1;
    loadedModel1 = osgDB::readNodeFile("../broom.obj");
    optimizer.optimize(loadedModel1.get());
    loadedModel1->accept(smooth);

    //Modification de l'objet
```

```
    osg::ref_ptr<PoLAR::Object3D> obj1 = new PoLAR::Object3D(loadedModel1.get(), true,
true);
    obj1->scale(30.0f);
    obj1->translate(10,-10,0);
    obj1->rotate(M_PI, 0, 1, 0);

    //Chargement de la matrice de projection
    PoLAR::Viewer::ProjectionType pt=PoLAR::Viewer2D3D::VISION;
    osg::Matrixd P = PoLAR::Util::readProjectionMatrix("../matrix.txt");
    viewer.setProjection(P, pt);

    //Création de l'objet invisible pour l'ombre et cacher le balai
    osg::ref_ptr<osg::Geode> tableGeode = new osg::Geode;
    osg::ref_ptr<osg::ShapeDrawable> shape;
    shape = new osg::ShapeDrawable(new osg::Box(osg::Vec3(25.0f, 12.5f, 0.0f), 50.0f,
25.0f, 0.00001f));
    tableGeode->addDrawable(shape.get());
    osg::ref_ptr<PoLAR::Object3D> object3D = new PoLAR::Object3D(tableGeode.get());
    object3D->setName("Table");
    object3D->setPhantomOn();


    //Ajout des objets et source de lumière
    viewer.addLightSource(20,15, -9, true);
    viewer.addObject3D(obj2.get());
    viewer.setTrackNode(obj2.get());
    viewer.addObject3D(obj1.get());
    viewer.setTrackNode(obj1.get());
    viewer.addObject3D(object3D.get());

    // Show the widget
    viewer.show();

    //Assure la fermeture correcte de la fenêtre
    app.connect( &app, SIGNAL(lastWindowClosed()), &app, SLOT(quit()) );
    // Exécute l'application
    return app.exec();
}
```

Annexe C : Photo utilisée

Annexe D : Résultat final