PixelToPosEstimator

P. Baillehache

December 2, 2018

Contents

Introduction

PixelToPosEstimator is a C library providing structures and functions to estimate 3D position from a 2D image.

The problem is as follow: given the 2D image of a set of points laying on a plane seen from a point of view, if we know the 3D position of the point of view and the 3D position and 2D position in the image of at least 3 points on the plane, how to estimate the 3D position of other points from their 2D position in the image.

PixelToPosEstimator provides a solution by using the known points to search the parameters optimizing a model of the projection from the 3D coordinates to the 2D coordinates, and then uses this optimized model to calculate other points. The optimization is performed using the `GenAlg` library.

It uses the `PBErr`, `PBMath`, `GSet`, `GenAlg` libraries.

1 Definitions

The problem consists of finding the function $F()$ such as $\overrightarrow{R_p} = F(\overrightarrow{S_p})$, where $\overrightarrow{R_p}$ is the 3D coordinates of the point p and $\overrightarrow{S_p}$ is the screen coordinates of the point p, shorten below as, respectively, $\overrightarrow{R}$, $\overrightarrow{S}$.

Screen coordinates system $(\overrightarrow{0}, \overrightarrow{u}, \overrightarrow{v})$ is as follow: $\overrightarrow{0}$ is at the top-left corner, $\overrightarrow{u}$ is toward the right and $\overrightarrow{v}$ is toward the bottom. Screen dimensions are noted W for width and H for height.

Lets define $\overrightarrow{P_p}$, the "polar" coordinates of the point p, shorten below as $\overrightarrow{P}$, as follow:

$$\overrightarrow{P} = \begin{cases} \frac{S_u - 0.5W}{0.5W} \\ \frac{S_v - 0.5H}{0.5H} \end{cases} \tag{1}$$

The polar coordinates represents the deviation of the point from the center of the screen relatively to the screen dimensions.

Lets now consider the projection of the screen on $\mathcal{S}$ the unit sphere centered on the camera position $\overrightarrow{C}$. The camera's orthonormal coordinates system is $(\overrightarrow{C}, \overrightarrow{right}, \overrightarrow{up}, \overrightarrow{depth})$ such as $\overrightarrow{depth}$ is colinear to $\overrightarrow{CV}$, where $\overrightarrow{V}$ is the point of view of the camera.

if we note α and β the angle of view of the camera along respectively $\overrightarrow{x}$ and $\overrightarrow{y}$, $\overrightarrow{P_\mathcal{S}}$ the projection of $\overrightarrow{S}$ on $\mathcal{S}$ is calculated as follow:

$$\overrightarrow{P'_\mathcal{S}} = Rot_{\overrightarrow{up}}(\overrightarrow{depth}, \alpha P_x) + Rot_{\overrightarrow{right}}(\overrightarrow{depth}, \beta P_y) - \overrightarrow{depth} \tag{2}$$

$$\overrightarrow{P_\mathcal{S}} = \frac{\overrightarrow{P'_\mathcal{S}}}{||\overrightarrow{P'_\mathcal{S}}||} \tag{3}$$

where $Rot_{\overrightarrow{w}}(\overrightarrow{A}, \theta)$ is the right-handed rotation of $\overrightarrow{A}$ around $\overrightarrow{w}$ by θ (in radians). We remind that the rotation matrix M is equal to (to shorten notation

θ is not written in the matrix below):

$$M = \begin{bmatrix} cos + w_x^2(1-cos) & w_xw_y(1-cos) - w_zsin & w_xw_z(1-cos) + w_ysin \\ w_xw_y(1-cos) + w_zsin & cos + w_y^2(1-cos) & w_yw_z(1-cos) - w_xsin \\ w_xw_z(1-cos) - w_ysin & w_yw_z(1-cos) + w_xsin & cos + w_z^2(1-cos) \end{bmatrix} \tag{4}$$

From $\overrightarrow{P_\mathcal{S}}$ it is then possible to calculate $\overrightarrow{R}$ as follow: it is the intersection of the line $(CP_\mathcal{S})$ and the plane $(\overrightarrow{x}, \overrightarrow{z})$:

$$\overrightarrow{R} = (\overrightarrow{C} + \gamma\overrightarrow{CP_\mathcal{S}})\Big|_{C_y+\gamma(P_{\mathcal{S}y}-C_y)=0} \tag{5}$$

which gives us the expression of the searched function $F()$.

However, this function relies on several unknown values:

- $\overrightarrow{V}$ the point of view of the camera
- α and β the angle of view of the camera
- $\overrightarrow{up}$ the up direction in the camera coordinates system. ($\overrightarrow{right}$ is not an unknown as we can calculate it as follow: $\overrightarrow{right} = \overrightarrow{depth} * \overrightarrow{up}$)

Then, given $\mathcal{I}$ the set of points for which we know both the 3D coordinates and 2D coordinates, we approximate these unknown values by solving the minimization problem:

$$\begin{cases} V_x, V_y, V_z, \alpha, \beta, up_x, up_y, up_z \in \mathbb{R}^8 \\ Min_{(\mathcal{I},\overrightarrow{V},\alpha,\beta,\overrightarrow{up})}(\frac{1}{\mathcal{I}^*}\sum_{i\in\mathcal{I}}||\overrightarrow{R_i} - F_{(\overrightarrow{V},\alpha,\beta,\overrightarrow{up})}(\overrightarrow{S_i})||) \end{cases} \tag{6}$$

2 Interface

```
// ============ PIXELTOPOSESTIMATOR.H ================

#ifndef PIXELTOPOSESTIMATOR_H
#define PIXELTOPOSESTIMATOR_H

// ================= Include =================

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdbool.h>
#include "pberr.h"
#include "pbmath.h"
#include "gset.h"
```

```
#include "genalg.h"

// ================= Define ==================

#define PTPE_Px(that) VecGet(that->_param, 0)
#define PTPE_Py(that) VecGet(that->_param, 1)
#define PTPE_Pz(that) VecGet(that->_param, 2)
#define PTPE_Sx(that) VecGet(that->_param, 3)
#define PTPE_Sy(that) VecGet(that->_param, 4)
#define PTPE_Upx(that) VecGet(that->_param, 5)
#define PTPE_Upy(that) VecGet(that->_param, 6)
#define PTPE_Upz(that) VecGet(that->_param, 7)

#define PTPE_NBPARAM 8

// ------------ PixelToPosEstimator

// ================= Data structure ===================

typedef struct PixelToPosEstimator {
  // Camera position
  VecFloat3D _cameraPos;
  // Dimension of the image
  VecFloat2D _imgSize;
  // Projection parameters
  // (Px, Py, Pz, Sx, Sy, Upx, Upy, Upz)
  VecFloat* _param;
} PixelToPosEstimator;

// ================ Functions declaration ====================

// Create a new PixelToPosEstimator
PixelToPosEstimator PixelToPosEstimatorCreateStatic(
  VecFloat3D* posCamera, const VecFloat2D* const imgSize);

// Free memory used by the PixelToPosEstimator 'that'
void PixelToPosEstimatorFreeStatic(PixelToPosEstimator* that);

// Convert the screen position to a polar position
VecFloat2D PTPEGetPxToPolar(
  const PixelToPosEstimator* const that,
  const VecFloat2D* const screenPos);

// Calculate the projection parameter using genetic algorithm for
// 'nbEpoch' epochs or until the average error gets below 'prec'
// Search for the parameters Px, Py, Pz in the bounding box defined
// by POVmin-POVmax
// the random generator must be initialized before calling this function
void PTPEInit(PixelToPosEstimator* const that,
  const GSet* const posMeter, const GSet* const posPixel,
  const unsigned int nbEpoch, const float prec,
  const VecFloat3D* const POVmin, const VecFloat3D* const POVmax);

// Convert the screen position to a real position
VecFloat3D PTPEGetPxToMeter(
  const PixelToPosEstimator* const that,
  const VecFloat2D* const screenPos);

// Convert the polar position to a real position
VecFloat3D PTPEGetPolarToMeter(
  const PixelToPosEstimator* const that,
  const VecFloat2D* const polarPos);
```

```
#endif
```

3 Code

3.1 pixeltoposestimator.c

```
#include "pixeltoposestimator.h"

// Create a new PixelToPosEstimator
PixelToPosEstimator PixelToPosEstimatorCreateStatic(
  VecFloat3D* posCamera, const VecFloat2D* const imgSize) {
#if BUILDMODE == 0
  if (posCamera == NULL) {
    ELORankErr->_type = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg, "'posCamera' is null");
    PBErrCatch(PixelToPosEstimatorErr);
  }
  if (imgSize == NULL) {
    ELORankErr->_type = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg, "'imgSize' is null");
    PBErrCatch(PixelToPosEstimatorErr);
  }
#endif
  // Declare the new estimator
  PixelToPosEstimator estimator;
  // Init the estimator
  estimator._cameraPos = *posCamera;
  estimator._imgSize = *imgSize;
  estimator._param = VecFloatCreate(PTPE_NBPARAM);
  // Return the new estimator
  return estimator;
}

// Free memory used by the PixelToPosEstimator 'that'
void PixelToPosEstimatorFreeStatic(PixelToPosEstimator* const that) {
  if (that == NULL)
    return;
  VecFree(&(that->_param));
}

// Convert the polar position to a real position
VecFloat3D PTPEGetPolarToMeter(
  const PixelToPosEstimator* const that,
  const VecFloat2D* const polarPos) {
#if BUILDMODE == 0
  if (that == NULL) {
    ELORankErr->_type = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg, "'that' is null");
    PBErrCatch(PixelToPosEstimatorErr);
  }
  if (polarPos == NULL) {
    ELORankErr->_type = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg, "'polarPos' is null");
    PBErrCatch(PixelToPosEstimatorErr);
  }
#endif
  // Declare a variable to memorize the result
  VecFloat3D res = VecFloatCreateStatic3D();
  // Calculate the real coordinates

  // POV
  VecFloat3D P = VecFloatCreateStatic3D();
  VecSet(&P, 0, PTPE_Px(that));
  VecSet(&P, 1, PTPE_Py(that));
  VecSet(&P, 2, PTPE_Pz(that));
  // Normalized vector Camera->POV
  VecFloat3D CP = VecGetOp(&P, 1.0, &(that->_cameraPos), -1.0);
  VecNormalise(&CP);
  // Normalized up vector
  VecFloat3D Up = VecFloatCreateStatic3D();
  VecSet(&Up, 0, PTPE_Upx(that));
  VecSet(&Up, 1, PTPE_Upy(that));
  VecSet(&Up, 2, PTPE_Upz(that));
  VecNormalise(&Up);
  // Normalized right vector
  VecFloat3D Right = VecCrossProd(&CP, &Up);
  VecNormalise(&Right);
  // Rotation according to up and right
  VecFloat3D Rx = CP;
  VecRotAxis(&Rx, &Up, PTPE_Sx(that) * VecGet(polarPos, 0));
  Rx = VecGetOp(&Rx, 1.0, &CP, -1.0);
  VecFloat3D Ry = CP;
  VecRotAxis(&Ry, &Right, PTPE_Sy(that) * VecGet(polarPos, 1));
  Ry = VecGetOp(&Ry, 1.0, &CP, -1.0);
  // 3d vector from camera corresponding to the polar pos pixel
  VecFloat3D V = VecGetOp(&CP, 1.0, &Rx, 1.0);
  V = VecGetOp(&V, 1.0, &Ry, 1.0);
  VecNormalise(&V);
  // Projection to ground plane
  float a = VecGet(&(that->_cameraPos), 1) / VecGet(&V, 1);
  VecSet(&res, 0, VecGet(&(that->_cameraPos), 0) - a * VecGet(&V, 0));
  VecSet(&res, 1, 0.0);
  VecSet(&res, 2, VecGet(&(that->_cameraPos), 2) - a * VecGet(&V, 2));

  // Return the result
  return res;
}

// Convert the screen position to a polar position
VecFloat2D PTPEGetPxToPolar(
  const PixelToPosEstimator* const that,
  const VecFloat2D* const screenPos) {
#if BUILDMODE == 0
  if (that == NULL) {
    ELORankErr->_type = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg, "'that' is null");
    PBErrCatch(PixelToPosEstimatorErr);
  }
  if (screenPos == NULL) {
    ELORankErr->_type = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg, "'screenPos' is null");
    PBErrCatch(PixelToPosEstimatorErr);
  }
#endif
  // Declare a variable to memorize the result
  VecFloat2D res = VecFloatCreateStatic2D();
  // Calculate the polar coordinates
  float w = VecGet(&(that->_imgSize), 0);
  VecSet(&res, 0, (VecGet(screenPos, 0) - 0.5 * w) / (0.5 * w));
  float h = VecGet(&(that->_imgSize), 1);
  VecSet(&res, 1, (VecGet(screenPos, 1) - 0.5 * h) / (0.5 * h));
  // Return the result
  return res;
}

// Calculate the projection parameter using genetic algorithm for
// 'nbEpoch' epochs or until the average error gets below 'prec'
// Search for the parameters Px, Py, Pz in the bounding box defined
// by POVmin-POVmax
// the random generator must be initialized before calling this function
void PTPEInit(PixelToPosEstimator* const that,
  const GSet* const posMeter, const GSet* const posPixel,
  const unsigned int nbEpoch, const float prec,
  const VecFloat3D* const POVmin, const VecFloat3D* const POVmax) {
#if BUILDMODE == 0
  if (that == NULL) {
    ELORankErr->_type = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg, "'that' is null");
    PBErrCatch(PixelToPosEstimatorErr);
  }
  if (posMeter == NULL) {
    ELORankErr->_type = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg, "'posMeter' is null");
    PBErrCatch(PixelToPosEstimatorErr);
  }
  if (posPixel == NULL) {
    ELORankErr->_type = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg, "'posPixel' is null");
    PBErrCatch(PixelToPosEstimatorErr);
  }
  if (GSetNbElem(posPixel) != GSetNbElem(posMeter)) {
    ELORankErr->_type = PBErrTypeInvalidArg;
    sprintf(ELORankErr->_msg,
      "'posPixel' and 'posMeter' don't have same sizes (%ld==%ld)",
      GSetNbElem(posPixel), GSetNbElem(posMeter));
    PBErrCatch(PixelToPosEstimatorErr);
  }
  if (POVmin == NULL) {
    ELORankErr->_type = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg, "'POVmin' is null");
    PBErrCatch(PixelToPosEstimatorErr);
  }
  if (POVmax == NULL) {
    ELORankErr->_type = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg, "'POVmax' is null");
    PBErrCatch(PixelToPosEstimatorErr);
  }
#endif
  // Create the GenAlg
  int lengthAdnF = PTPE_NBPARAM;
  int lengthAdnI = 0;
  GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
    lengthAdnF, lengthAdnI);
  // Set the boundaries for the parameters
  VecFloat2D boundsF = VecFloatCreateStatic2D();
  VecSet(&boundsF, 0, VecGet(POVmin, 0));
  VecSet(&boundsF, 1, VecGet(POVmax, 0));
  GASetBoundsAdnFloat(ga, 0, &boundsF); // Px
  VecSet(&boundsF, 0, VecGet(POVmin, 1));
  VecSet(&boundsF, 1, VecGet(POVmax, 1));
  GASetBoundsAdnFloat(ga, 1, &boundsF); // Py
```

```
VecSet(&boundsF, 0, VecGet(POVmin, 2));
VecSet(&boundsF, 1, VecGet(POVmax, 2));
GASetBoundsAdnFloat(ga, 2, &boundsF); // Pz
VecSet(&boundsF, 0, -PBMATH_HALFPI);
VecSet(&boundsF, 1, PBMATH_HALFPI);
GASetBoundsAdnFloat(ga, 3, &boundsF); // Sx
GASetBoundsAdnFloat(ga, 4, &boundsF); // Sy
VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
GASetBoundsAdnFloat(ga, 5, &boundsF); // Upx
VecSet(&boundsF, 0, 0.0); VecSet(&boundsF, 1, 1.0);
GASetBoundsAdnFloat(ga, 6, &boundsF); // Upy
VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
GASetBoundsAdnFloat(ga, 7, &boundsF); // Upz
// Init the GenAlg
GAInit(ga);
// Variable to memorize the current best adn value
float best = 10000.0;
// Loop on epochs
do {
  //printf("epoch %ld avg err %fm     \r",
  //  GAGetCurEpoch(ga), best / (float)GSetNbElem(posMeter));
  //fflush(stdout);
  // Variable to memorize the evaluation of one base
  float ev = 0.0;
  // Loop on adns
  for (int iEnt = 0; iEnt < GAGetNbAdns(ga); ++iEnt) {
    // Copy the adn into the estimator's parameters
    VecCopy(that->_param, GAAdnAdnF(GAAdn(ga, iEnt)));
    // Reset the evaluation variable
    ev = 0.0;
    // Loop on both sets
    GSetIterForward iterMeter = GSetIterForwardCreateStatic(posMeter);
    GSetIterForward iterPixel = GSetIterForwardCreateStatic(posPixel);
    do {
      // Get the screen position
      VecFloat2D* pPixel = GSetIterGet(&iterPixel);
      // Convert to polar position
      VecFloat2D polarPos = PTPEGetPxToPolar(that, pPixel);
      // Convert to real position
      VecFloat3D pEstim = PTPEGetPolarToMeter(that, &polarPos);
      // Get the correct real position
      VecFloat3D* pMeter = GSetIterGet(&iterMeter);
      // Calculate the error
      ev += VecDist(pMeter, &pEstim);
    } while (GSetIterStep(&iterMeter) && GSetIterStep(&iterPixel));
    // Calculate the average error
    ev /=  (float)GSetNbElem(posMeter);
    // Update the value of this adn
    GASetAdnValue(ga, GAAdn(ga, iEnt), -1.0 * ev);
    // Update the best value if necessary
    if (ev < best - PBMATH_EPSILON) {
      best = ev;
      printf("%lu %f ", GAGetCurEpoch(ga), best);
      VecFloatPrint(that->_param, stdout, 6);
      printf("        \n"); fflush(stdout);
    }
  }
  // Step the GenAlg
  GAStep(ga);
} while (GAGetCurEpoch(ga) < nbEpoch && best > prec);
// Copy the final best adn into the estimator's parameters
VecCopy(that->_param, GAAdnAdnF(GABestAdn(ga)));
```

```
  // Free memory
  GenAlgFree(&ga);
}

// Convert the screen position to a real position
VecFloat3D PTPEGetPxToMeter(
  const PixelToPosEstimator* const that,
  const VecFloat2D* const screenPos) {
#if BUILDMODE == 0
  if (that == NULL) {
    ELORankErr->_type = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg, "'that' is null");
    PBErrCatch(PixelToPosEstimatorErr);
  }
  if (screenPos == NULL) {
    ELORankErr->_type = PBErrTypeNullPointer;
    sprintf(ELORankErr->_msg, "'screenPos' is null");
    PBErrCatch(PixelToPosEstimatorErr);
  }
#endif
  // Declare a variable to memorize the result
  VecFloat3D res = VecFloatCreateStatic3D();
  // Calculate the real coordinates
  VecFloat2D polarPos = PTPEGetPxToPolar(that, screenPos);
  res = PTPEGetPolarToMeter(that, &polarPos);
  // Return the result
  return res;
}
```

4 Makefile

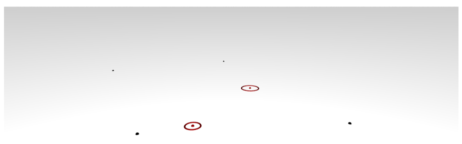
```
# Build mode
# 0: development (max safety, no optimisation)
# 1: release (min safety, optimisation)
# 2: fast and furious (no safety, optimisation)
BUILD_MODE?=1

all: pbmake_wget main ground.png

# Automatic installation of the repository PBMake in the parent folder
pbmake_wget:
if [ ! -d ../PBMake ]; then wget https://github.com/BayashiPascal/PBMake/archive/master.zip; unzip master.zip; rm -f

# Makefile definitions
MAKEFILE_INC=../PBMake/Makefile.inc
include $(MAKEFILE_INC)

# Rules to make the executable
repo=pixeltoposestimator
$($(repo)_EXENAME): \
$($(repo)_EXENAME).o \
$($(repo)_EXE_DEP) \
$($(repo)_DEP)
$(COMPILER) `echo "$($(repo)_EXE_DEP) $($(repo)_EXENAME).o" | tr ' ' '\n' | sort -u` $(LINK_ARG) $($(repo)_LINK_ARG)

$($(repo)_EXENAME).o: \
$($(repo)_DIR)/$($(repo)_EXENAME).c \
$($(repo)_INC_H_EXE) \
$($(repo)_EXE_DEP)
$(COMPILER) $(BUILD_ARG) $($(repo)_BUILD_ARG) ‘echo "$($(repo)_INC_DIR)" | tr ’ ’ ’\n’ | sort -u‘ -c $($(repo)_DIR)/

ground.png: ground.pov
povray -W1280 -H720 -P -Q9 +A -Iground.pov
```

5 Example

5.1 Test case

```
#include "colors.inc"
#include "textures.inc"

background {
color White
}

#declare CameraPos = <0, 15, 0>;
#declare POV = <50, 0, 50>;
camera {
location CameraPos
look_at POV
  up y
  right x * 1280 / 720
}

light_source {
CameraPos
White
}

plane {
  y, 0
  texture {
    pigment { color rgb 0.9 }
    finish { ambient 0.9 }
  }
}

sphere {
  <48,0,51>
  .2
  pigment { color Black }
}
sphere {
  <12,0,28>
  .2
  pigment { color Black }
}
sphere {
  <33,0,11>
  .2
  pigment { color Black }
}
sphere {
  <22,0,62>
  .2
  pigment { color Black }
}

sphere {
  <18,0,25>
  .2
  pigment { color Red }
}
torus {
  1.02698, .1
  translate <18,0,25>
  pigment { color Red }
}
sphere {
  <35,0,30>
  .2
  pigment { color Red }
}
torus {
  1.601299, .1
  translate <35,0,30>
  pigment { color Red }
}
```

In the image below, the four input positions are in black, and the 2 test positions are in red. The red circles show the error range of the estimation.:



5.2 main.c

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <tgmath.h>
#include "pixeltoposestimator.h"


int main(int argc, char** argv) {
  (void)argc; (void)argv;

  srandom(time(NULL));

  // Read the data from the file in argument
  if (argc != 2) {
```

```
  fprintf(stderr, "Usage: main <input file>\n");
  exit(0);
}
FILE* inputFile = fopen(argv[1], "r");
if (inputFile == NULL) {
  fprintf(stderr, "Can't open %s\n", argv[0]);
  exit(0);
}
VecFloat* posCamera = NULL;
VecFloat* imgSize = NULL;
VecLoad(&posCamera, inputFile);
VecLoad(&imgSize, inputFile);
VecFloat* POVmin = NULL;
VecFloat* POVmax = NULL;
VecLoad(&POVmin, inputFile);
VecLoad(&POVmax, inputFile);
int nbInput = 0;
int ret = fscanf(inputFile, "%d", &nbInput);
GSet inputMeter = GSetCreateStatic();
GSet inputPixel = GSetCreateStatic();
for (int iInput = 0; iInput < nbInput; ++iInput) {
  VecFloat* posMeter = NULL;
  VecFloat* posPixel = NULL;
  VecLoad(&posMeter, inputFile);
  VecLoad(&posPixel, inputFile);
  GSetAppend(&inputMeter, posMeter);
  GSetAppend(&inputPixel, posPixel);
}
int nbTest = 0;
ret = fscanf(inputFile, "%d", &nbTest);
(void)ret;
GSet testMeter = GSetCreateStatic();
GSet testPixel = GSetCreateStatic();
for (int iInput = 0; iInput < nbTest; ++iInput) {
  VecFloat* posMeter = NULL;
  VecFloat* posPixel = NULL;
  VecLoad(&posMeter, inputFile);
  VecLoad(&posPixel, inputFile);
  GSetAppend(&testMeter, posMeter);
  GSetAppend(&testPixel, posPixel);
}
fclose(inputFile);

// Create the estimator
PixelToPosEstimator estimator = PixelToPosEstimatorCreateStatic(
  (VecFloat3D*)posCamera, (VecFloat2D*)imgSize);

// Calculate the projection parameters
FILE* fileParam = fopen("./param.txt", "r");
if (fileParam == NULL) {
  printf("Calculate the projection param...\n");
  unsigned int nbEpoch = 500000;
  float prec = 0.001;
  PTPEInit(&estimator, &inputMeter, &inputPixel,
    nbEpoch, prec, (VecFloat3D*)POVmin, (VecFloat3D*)POVmax);
  fileParam = fopen("./param.txt", "w");
  if (!VecSave(estimator._param, fileParam, true)) {
    fprintf(stderr, "Failed to save the parameters\n");
    exit(0);
  }
} else {
  printf("Reuse the projection param...\n");
  if (!VecLoad(&(estimator._param), fileParam)) {
    fprintf(stderr, "Failed to load the parameters\n");
    exit(0);
  }
}
fclose(fileParam);

printf("\n");
printf("Projection param: ");
VecPrint(estimator._param, stdout);printf("\n");
printf("\n");

printf("Input data:\n\n");
float avgErr = 0.0;
float maxErr = 0.0;
for (int iInput = 0; iInput < nbInput; ++iInput) {
  VecFloat3D* posMeter = (VecFloat3D*)GSetGet(&inputMeter, iInput);
  VecFloat2D* posPixel = (VecFloat2D*)GSetGet(&inputPixel, iInput);
  printf("input #%d (m): ", iInput);
  VecPrint(posMeter, stdout);
  printf(" (px): ");
  VecPrint(posPixel, stdout);
  printf("\n");
  printf(" (screen->real): ");
  VecFloat3D estimPos = PTPEGetPxToMeter(&estimator, posPixel);
  VecPrint(&estimPos, stdout);
  float error = VecDist(&estimPos, posMeter);
  printf(" (error): %fm", error);
  printf("\n\n");
  avgErr += error;
  if (maxErr < error)
    maxErr = error;
}
avgErr /= (float)nbInput;
printf("Average error: %fm\n", avgErr);
printf("Max error: %fm\n\n", maxErr);

printf("Test data:\n\n");
avgErr = 0.0;
maxErr = 0.0;
for (int iInput = 0; iInput < nbTest; ++iInput) {
  VecFloat3D* posMeter = (VecFloat3D*)GSetGet(&testMeter, iInput);
  VecFloat2D* posPixel = (VecFloat2D*)GSetGet(&testPixel, iInput);
  printf("input #%d (m): ", iInput);
  VecPrint(posMeter, stdout);
  printf(" (px): ");
  VecPrint(posPixel, stdout);
  printf("\n");
  printf(" (screen->real): ");
  VecFloat3D estimPos = PTPEGetPxToMeter(&estimator, posPixel);
  VecPrint(&estimPos, stdout);
  float error = VecDist(&estimPos, posMeter);
  printf(" (error): %fm", error);
  printf("\n\n");
  avgErr += error;
  if (maxErr < error)
    maxErr = error;
}
avgErr /= (float)nbTest;
printf("Average error: %fm\n", avgErr);
printf("Max error: %fm\n", maxErr);
```

```
  // Free memory
  PixelToPosEstimatorFreeStatic(&estimator);
  while (GSetNbElem(&inputMeter) > 0) {
    VecFloat* v = GSetPop(&inputMeter);
    VecFree(&v);
  }
  while (GSetNbElem(&inputPixel) > 0) {
    VecFloat* v = GSetPop(&inputPixel);
    VecFree(&v);
  }
  while (GSetNbElem(&testMeter) > 0) {
    VecFloat* v = GSetPop(&testMeter);
    VecFree(&v);
  }
  while (GSetNbElem(&testPixel) > 0) {
    VecFloat* v = GSetPop(&testPixel);
    VecFree(&v);
  }

  // Return success code
  return 0;
}
```

inputTest.txt:

```
{
  "_dim":"3",
  "_val":["0.0","10.0","0.0"]
}
{
  "_dim":"2",
  "_val":["1280.000000","720.000000"]
}
{
  "_dim":"3",
  "_val":["0.0","0.0","0.0"]
}
{
  "_dim":"3",
  "_val":["10.0","15.0","10.0"]
}
4
{
  "_dim":"3",
  "_val":["48.000000","0.000000","51.000000"]
}
{
  "_dim":"2",
  "_val":["618.000000","361.000000"]
}
{
  "_dim":"3",
  "_val":["12.000000","0.000000","28.000000"]
}
{
  "_dim":"2",
  "_val":["375.000000","565.000000"]
}
{
  "_dim":"3",
  "_val":["33.000000","0.000000","11.000000"]
}
{
  "_dim":"2",
  "_val":["974.000000","536000000"]
}
{
  "_dim":"3",
  "_val":["22.000000","0.000000","62.000000"]
}
{
  "_dim":"2",
  "_val":["307.000000","387.000000"]
}
2
{
  "_dim":"3",
  "_val":["18.000000","0.000000","25.000000"]
}
{
  "_dim":"2",
  "_val":["531.000000","543.000000"]
}
{
  "_dim":"3",
  "_val":["35.000000","0.000000","30.000000"]
}
{
  "_dim":"2",
  "_val":["692.000000","436.000000"]
}
```

6 Output

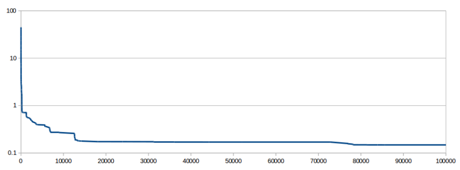
```
Calculate the projection param...
0 44.884600 <6.281383,2.544475,4.045629,-0.810402,-1.440463,-0.251248,0.821694,0.853980>
0 44.406651 <3.035180,6.493492,8.269773,-0.487826,0.523906,0.314290,0.622056,0.278442>
0 39.926086 <5.686362,4.805532,4.719906,-0.361895,-1.535800,-0.391348,0.965750,-0.972703>
0 28.386356 <9.420714,8.037761,8.992093,0.081077,-1.201721,0.260587,0.531561,0.954478>
1 23.291659 <8.907933,8.036860,7.472357,-0.261465,0.341481,-0.964939,0.036744,-0.757546>
1 23.157566 <9.765271,8.092203,8.723720,-0.329725,-0.585552,0.446517,0.302049,-0.367535>
1 18.792313 <9.818071,8.036860,7.472357,1.318501,-0.356141,0.260587,0.531561,0.495598>
2 18.077366 <9.765271,8.092203,8.738153,0.081077,-0.585552,0.446517,0.531561,-0.376360>
3 17.387600 <3.472870,8.037761,8.994494,-0.157184,-0.585552,-0.469345,0.531561,0.954478>
3 15.448114 <9.420714,8.092203,8.723720,0.084058,-0.891022,0.260587,0.531561,-0.376360>
4 14.997634 <8.924720,8.092203,8.723720,0.084058,-0.782852,0.260587,0.342059,-0.367535>
4 12.908067 <9.818071,8.036860,9.793681,1.318501,-0.356141,0.260587,0.402870,0.495598>
7 8.401557 <9.818071,8.036860,9.793681,1.188305,-0.356141,0.345925,0.402870,0.495598>
12 7.753658 <9.420714,8.092203,9.730992,0.775250,-0.891022,0.533777,0.911558,0.495598>
13 7.675824 <9.420714,8.092203,9.781833,0.707945,-0.524464,0.406542,0.770881,0.511776>
13 7.159789 <8.924720,8.036860,9.793681,1.044841,-0.524464,0.315572,0.770881,0.495598>
13 6.693418 <9.439643,8.092203,9.730992,0.775250,-0.524464,0.446517,0.855537,0.495598>
14 6.400322 <9.420714,8.092203,8.480831,1.111742,-0.524464,0.406542,0.770881,0.511776>
15 4.594604 <9.818071,8.092203,9.730992,1.184015,-0.524464,0.345925,0.402870,0.495598>
20 3.776099 <9.420714,8.092203,9.793681,1.184015,-0.524464,0.345925,0.484037,0.495598>
24 3.525790 <9.420714,8.092203,9.628260,1.184015,-0.524464,0.345925,0.484037,0.495598>
30 3.258011 <9.604491,8.092203,9.866126,1.111742,-0.524464,0.406542,0.573255,0.582437>
32 3.117882 <9.937201,8.092203,9.919612,1.111742,-0.524464,0.345925,0.573255,0.495598>
34 3.117851 <9.937201,8.092203,9.919612,1.111742,-0.524464,0.406542,0.646383,0.582437>
```

```
35 2.938462 <9.937201,8.092203,9.919612,1.184015,-0.524464,0.406542,0.560358,0.582437>
48 2.818095 <9.681943,8.092203,9.972069,1.142336,-0.524464,0.406542,0.561197,0.582437>
55 2.797715 <9.604491,8.092203,9.919612,1.142336,-0.524464,0.406542,0.540557,0.582437>
58 2.720654 <9.681943,8.092203,9.919612,1.142336,-0.524464,0.404598,0.573356,0.582437>
82 2.701498 <9.777955,8.092203,9.972069,1.136984,-0.524464,0.406542,0.557293,0.582437>
102 2.476554 <9.621267,8.092203,9.919612,1.142336,-0.415372,0.406542,0.534199,0.582437>
107 2.222970 <9.621267,8.092203,9.919612,1.142336,-0.415372,0.406542,0.534199,0.606424>
111 2.177721 <9.777955,8.092203,9.955930,1.136984,-0.415372,0.406542,0.482520,0.582437>
113 2.126257 <9.785274,8.092203,9.919612,1.142336,-0.415372,0.406542,0.561197,0.587305>
113 2.050461 <9.621267,8.092203,9.919612,1.142336,-0.415372,0.406542,0.586852,0.606424>
114 1.925390 <9.777955,8.092203,9.919612,1.136984,-0.415372,0.406542,0.612972,0.606424>
142 1.905300 <9.621267,8.092203,9.897383,1.120821,-0.415372,0.406542,0.586852,0.606424>
148 1.839559 <9.777955,8.092203,9.955930,1.136984,-0.415372,0.406542,0.597292,0.606424>
149 1.818513 <9.689944,8.092203,9.955930,1.103658,-0.415372,0.406542,0.618273,0.606424>
170 1.786256 <9.763318,8.092203,9.955930,1.128148,-0.415372,0.406542,0.607047,0.606424>
182 1.580445 <9.777955,8.092203,9.919612,1.136984,-0.314057,0.406542,0.553064,0.606424>
183 1.538658 <9.915060,8.092203,9.955930,1.120821,-0.314057,0.406542,0.612972,0.606424>
187 1.400943 <9.787211,8.083845,9.897383,1.125060,-0.314057,0.406542,0.566475,0.606424>
189 1.336716 <9.584694,8.092203,9.890512,1.125060,-0.314057,0.406542,0.597292,0.620857>
191 1.195886 <9.645562,8.092203,9.897383,1.098414,-0.314057,0.406542,0.599253,0.606424>
192 1.078981 <9.631684,8.092203,9.890512,1.103517,-0.314057,0.406542,0.597292,0.620857>
195 1.026388 <9.631684,8.092203,9.943486,1.103517,-0.314057,0.406542,0.600996,0.620857>
200 1.006481 <9.606969,8.092203,9.890512,1.098414,-0.314057,0.406542,0.600996,0.620857>
204 0.923881 <9.748018,8.092203,9.897383,1.103517,-0.314057,0.406542,0.638712,0.620857>
216 0.922173 <9.652600,8.092203,9.888913,1.098414,-0.314057,0.406542,0.619672,0.620857>
222 0.763326 <9.713642,8.092203,9.943486,1.076891,-0.314057,0.406542,0.648829,0.620857>
244 0.756738 <9.675240,8.092203,9.888913,1.076891,-0.314057,0.406542,0.665544,0.620857>
252 0.751913 <9.695823,8.092203,9.897383,1.076891,-0.314057,0.406542,0.668927,0.620857>
270 0.751425 <9.742764,8.092203,9.897383,1.076891,-0.314057,0.406542,0.674066,0.620857>
273 0.746165 <9.695823,8.092203,9.897383,1.076891,-0.314057,0.406542,0.667074,0.620857>
285 0.739344 <9.752088,8.092203,9.923166,1.076891,-0.314057,0.406542,0.667074,0.620857>
296 0.731872 <9.773252,8.092203,9.923166,1.076891,-0.314057,0.406542,0.673085,0.620857>
339 0.730829 <9.817637,8.092203,9.923166,1.076891,-0.314057,0.406542,0.684769,0.620857>
361 0.721335 <9.784201,8.092203,9.923166,1.076891,-0.314057,0.406542,0.679696,0.620857>
370 0.721078 <9.784201,8.092203,9.923166,1.076891,-0.314057,0.406542,0.679010,0.620857>
460 0.720938 <9.792299,8.092203,9.923930,1.076891,-0.314057,0.406542,0.681922,0.620857>
491 0.718458 <9.792299,8.092203,9.923166,1.076014,-0.314057,0.406542,0.681922,0.620857>
539 0.713503 <9.786608,8.092203,9.923166,1.076891,-0.314057,0.406542,0.681922,0.621290>
604 0.712912 <9.792295,8.094030,9.923166,1.076014,-0.314057,0.406542,0.685137,0.621910>
736 0.712580 <9.769989,8.094030,9.923166,1.076014,-0.314057,0.406542,0.681122,0.621910>
1248 0.712495 <4.848133,9.036037,4.851320,0.864740,-0.336921,0.284323,0.975625,0.379136>
1249 0.643722 <4.848133,9.036037,4.862695,0.864740,-0.336921,0.284323,0.985079,0.373942>
1252 0.607572 <4.848133,9.036037,4.862695,0.864740,-0.336921,0.284323,0.994658,0.379136>
1258 0.600365 <4.848133,9.036037,4.862695,0.864740,-0.336921,0.284323,0.996807,0.379136>
1277 0.586545 <4.819316,9.036037,4.851320,0.868794,-0.336921,0.284323,0.999060,0.379136>
1295 0.586414 <4.819316,9.036037,4.851320,0.868794,-0.336921,0.284323,0.999099,0.379136>
1325 0.586258 <4.819316,9.036037,4.851320,0.868794,-0.336921,0.284323,0.999146,0.379136>
1363 0.577828 <4.844653,9.036037,4.866961,0.868794,-0.336921,0.284323,0.999060,0.379136>
1409 0.576681 <4.848133,9.036037,4.862695,0.868794,-0.336921,0.282206,0.998863,0.379136>
1452 0.565243 <4.844776,9.036037,4.862695,0.870687,-0.336921,0.282206,0.998863,0.379136>
1616 0.555914 <4.824059,9.036037,4.862695,0.868794,-0.336921,0.282206,0.999713,0.379136>
1793 0.555356 <4.831511,9.036037,4.862695,0.872947,-0.336921,0.282206,0.999713,0.379136>
1957 0.546738 <4.826589,9.036037,4.871719,0.870251,-0.336921,0.281504,0.999467,0.379136>
2126 0.533117 <4.830008,9.036037,4.862695,0.870251,-0.336921,0.279149,0.998679,0.376354>
2220 0.520408 <4.830008,9.036037,4.862695,0.870251,-0.336921,0.279149,0.999713,0.372581>
2259 0.519050 <4.845685,9.036037,4.871719,0.870251,-0.336921,0.279149,0.999467,0.373728>
2301 0.497474 <4.830008,9.036037,4.862695,0.870251,-0.336921,0.277030,0.999165,0.372581>
2419 0.494836 <4.819316,9.036037,4.862695,0.870843,-0.336921,0.277030,0.999165,0.372581>
2478 0.486439 <4.830008,9.036037,4.876121,0.870251,-0.336921,0.276127,0.999684,0.372581>
2583 0.478788 <4.845685,9.036037,4.862695,0.870251,-0.336921,0.273472,0.999467,0.368140>
2699 0.457038 <4.830290,9.036037,4.869759,0.870251,-0.336921,0.273472,0.997686,0.368140>
2784 0.456981 <4.830008,9.036037,4.862695,0.870251,-0.336921,0.273472,0.998643,0.365178>
```

```
2828 0.453112 <4.835212,9.036037,4.873294,0.870251,-0.336921,0.273472,0.999467,0.368140>
3318 0.433551 <8.511834,8.307278,8.570730,0.868274,-0.336921,0.268581,0.998242,0.363261>
3386 0.431667 <8.518552,8.307278,8.570730,0.869201,-0.336921,0.269664,0.999644,0.363261>
3394 0.426953 <8.495723,8.307278,8.558190,0.875499,-0.336921,0.268581,0.998242,0.363261>
3493 0.422279 <8.478710,8.307278,8.556923,0.874170,-0.336921,0.268581,0.999644,0.363261>
3500 0.420096 <8.467314,8.307278,8.570730,0.868274,-0.336921,0.268581,0.999735,0.363261>
3533 0.410834 <8.474053,8.307278,8.570730,0.869201,-0.336921,0.268581,0.999735,0.363261>
3567 0.405540 <8.489199,8.307278,8.570730,0.870533,-0.336921,0.268581,0.999644,0.363261>
3583 0.402555 <8.489199,8.307278,8.566431,0.870533,-0.336921,0.268581,0.999644,0.363261>
3667 0.401827 <8.478710,8.307278,8.570730,0.870533,-0.336921,0.267957,0.999447,0.363261>
3780 0.399194 <8.476549,8.307278,8.559493,0.870533,-0.336921,0.268581,0.999735,0.362369>
3887 0.396914 <8.476549,8.307278,8.551306,0.870533,-0.336921,0.268581,0.999735,0.362369>
4071 0.395418 <8.465826,8.307278,8.559493,0.870533,-0.336921,0.267957,0.999735,0.362369>
5327 0.391432 <8.479677,8.307278,8.560559,0.870533,-0.336921,0.267957,0.999532,0.361755>
5588 0.390964 <8.470735,8.307278,8.551970,0.870533,-0.336921,0.257371,0.960051,0.346775>
5594 0.372444 <8.498499,8.307278,8.565677,0.870533,-0.336921,0.257371,0.976702,0.346775>
5596 0.370464 <8.498499,8.307278,8.561835,0.870533,-0.336921,0.257371,0.987405,0.346775>
5598 0.367464 <8.498499,8.307278,8.557663,0.870533,-0.336921,0.257371,0.976333,0.346775>
5713 0.367187 <8.498499,8.307278,8.560231,0.870533,-0.336921,0.257371,0.977701,0.346775>
5756 0.366584 <8.503640,8.307278,8.561835,0.870533,-0.336921,0.257371,0.978313,0.346775>
6702 0.343021 <8.503640,8.307428,8.561835,0.865406,-0.327725,0.257371,0.987266,0.348621>
6729 0.337664 <8.503640,8.307428,8.561835,0.870010,-0.327725,0.257371,0.991874,0.346775>
6740 0.335527 <8.503640,8.307428,8.561835,0.870010,-0.327725,0.257371,0.992819,0.346775>
6756 0.329242 <8.503640,8.307428,8.561835,0.870010,-0.327725,0.257371,0.995828,0.346775>
6767 0.320022 <8.503640,8.307428,8.558463,0.870010,-0.327725,0.257371,0.997371,0.346775>
6790 0.317564 <8.503640,8.307428,8.561835,0.870010,-0.327725,0.257371,0.992819,0.349807>
6800 0.303163 <8.487885,8.307428,8.561835,0.870010,-0.327725,0.257371,0.995828,0.349807>
6830 0.290565 <8.504672,8.307278,8.561835,0.870010,-0.327725,0.255969,0.995442,0.347005>
6905 0.288653 <8.503640,8.306267,8.561835,0.870010,-0.327725,0.256058,0.997458,0.347444>
6933 0.286003 <8.503640,8.307428,8.558463,0.870010,-0.327725,0.256097,0.997140,0.347005>
6985 0.276318 <8.503542,8.306267,8.561835,0.870010,-0.327725,0.255969,0.999218,0.347005>
6992 0.276017 <8.487885,8.306267,8.561835,0.868395,-0.327725,0.255969,0.999218,0.347005>
7003 0.273886 <8.499309,8.306267,8.561835,0.870010,-0.327725,0.255969,0.999218,0.347005>
8856 0.273845 <8.517655,8.306267,8.553466,0.870010,-0.327725,0.047745,0.189622,0.064434>
8861 0.272887 <8.519454,8.306267,8.553466,0.870849,-0.327725,0.047745,0.189622,0.064434>
8935 0.271074 <8.517655,8.306267,8.550647,0.870849,-0.327725,0.047745,0.189622,0.064434>
9053 0.270190 <8.519454,8.306267,8.553212,0.870849,-0.327725,0.047745,0.191214,0.064434>
12521 0.259414 <8.167076,8.363256,8.225618,0.849739,-0.317070,0.113935,0.579978,0.146257>
12589 0.252231 <8.167076,8.363256,8.227421,0.849739,-0.317070,0.113935,0.590027,0.146257>
12594 0.250593 <8.214394,8.363256,8.225618,0.849739,-0.317070,0.113935,0.584331,0.146257>
12608 0.221227 <8.188238,8.363256,8.225618,0.853707,-0.317070,0.113935,0.579996,0.146257>
12629 0.213859 <8.214394,8.363256,8.225618,0.853707,-0.317070,0.113935,0.598188,0.146257>
12700 0.203871 <8.197347,8.363256,8.227421,0.853707,-0.317070,0.113935,0.597883,0.146257>
12723 0.202278 <8.197347,8.363256,8.227421,0.853707,-0.317070,0.113935,0.596273,0.146257>
12758 0.198890 <8.201456,8.363256,8.218951,0.853707,-0.317070,0.113935,0.597883,0.146257>
12761 0.189514 <8.201456,8.363256,8.218951,0.853540,-0.317070,0.113935,0.590497,0.146257>
12782 0.189494 <8.201456,8.363256,8.218951,0.853540,-0.317070,0.113935,0.590844,0.146257>
12845 0.188823 <8.201456,8.363256,8.221518,0.853707,-0.317070,0.113935,0.587777,0.146257>
13125 0.188811 <8.201456,8.363256,8.219174,0.854978,-0.317070,0.113935,0.595087,0.146257>
13168 0.187130 <8.203152,8.363256,8.219174,0.853707,-0.317070,0.113858,0.589913,0.146257>
13186 0.186052 <8.201456,8.363256,8.219174,0.853707,-0.317070,0.113858,0.588532,0.146257>
13284 0.185849 <8.201334,8.363256,8.222420,0.853707,-0.317070,0.113935,0.596706,0.146257>
13314 0.181894 <8.201456,8.363256,8.223605,0.853707,-0.317070,0.113858,0.592777,0.146257>
13458 0.181329 <8.201456,8.363256,8.223605,0.853707,-0.317070,0.113843,0.591444,0.146257>
13575 0.180988 <8.201456,8.363256,8.223605,0.853707,-0.317070,0.113843,0.592252,0.146257>
13648 0.180734 <8.201456,8.363256,8.223605,0.853707,-0.317070,0.113843,0.594153,0.146257>
13848 0.179969 <8.203213,8.363256,8.222116,0.853917,-0.317070,0.113843,0.594134,0.146257>
14071 0.178995 <8.203213,8.363256,8.222116,0.853917,-0.317070,0.113843,0.592235,0.146257>
14107 0.178980 <8.203213,8.363256,8.222116,0.853917,-0.317070,0.113843,0.592996,0.146257>
14234 0.178949 <8.203213,8.363256,8.222116,0.853917,-0.317070,0.113843,0.592522,0.146257>
18083 0.173994 <8.197326,8.363256,8.229916,0.852874,-0.317070,0.113450,0.585992,0.146140>
23891 0.173895 <8.913875,8.220592,8.939408,0.854255,-0.317070,0.131827,0.682465,0.169636>
```

```
31109 0.173203 <8.917038,8.220592,8.943800,0.853543,-0.317070,0.162205,0.839574,0.208871>
31204 0.173118 <8.917038,8.220592,8.943800,0.853543,-0.317070,0.162205,0.839853,0.208871>
31208 0.172188 <8.917038,8.220592,8.943800,0.853975,-0.317070,0.162205,0.842092,0.208871>
31291 0.171166 <8.917038,8.220592,8.943800,0.853975,-0.317070,0.162205,0.841311,0.208871>
36012 0.171123 <8.917038,8.220592,8.942698,0.853975,-0.317070,0.162205,0.839808,0.208871>
36092 0.170691 <8.917038,8.220592,8.943075,0.853975,-0.317070,0.162205,0.840325,0.208871>
44263 0.170482 <8.916946,8.220592,8.943075,0.853975,-0.317070,0.162205,0.840471,0.208871>
51927 0.170469 <8.916946,8.220592,8.943075,0.853975,-0.317070,0.162205,0.840493,0.208871>
72809 0.170448 <8.916946,8.220592,8.943075,0.853975,-0.317070,0.162205,0.840574,0.208871>
76934 0.158941 <8.960755,8.205441,8.954387,0.846258,-0.322978,0.129444,0.969470,0.145199>
76954 0.156855 <8.960755,8.205441,8.954387,0.846258,-0.322978,0.129444,0.967982,0.145199>
78037 0.153757 <9.115660,8.172626,9.122255,0.846486,-0.322978,0.128713,0.951663,0.145199>
78121 0.152142 <9.115660,8.172626,9.132977,0.848454,-0.322978,0.128713,0.964602,0.145199>
78127 0.151612 <9.118387,8.172626,9.122255,0.846486,-0.322978,0.128713,0.951663,0.145199>
78147 0.150881 <9.118387,8.172626,9.127856,0.846486,-0.322978,0.128713,0.960791,0.145199>
78218 0.150298 <9.118087,8.172626,9.132977,0.846486,-0.322978,0.128713,0.964069,0.145199>
78251 0.150018 <9.118387,8.172626,9.127856,0.846486,-0.322978,0.128713,0.959323,0.145199>
78271 0.149341 <9.118087,8.172626,9.132977,0.846486,-0.322978,0.128713,0.959483,0.145199>
81389 0.149188 <5.103513,8.977372,5.110988,0.845284,-0.322978,0.128713,0.956514,0.145263>
81406 0.148699 <5.103868,8.977372,5.109262,0.846719,-0.322978,0.128713,0.959184,0.145263>
81690 0.148686 <5.103868,8.977372,5.109262,0.846719,-0.322978,0.128713,0.958237,0.145263>
81921 0.148642 <5.103868,8.977372,5.109262,0.846496,-0.322978,0.128713,0.957903,0.145263>
83919 0.148591 <5.103868,8.977372,5.109262,0.845611,-0.322978,0.128713,0.954664,0.145263>
85583 0.148534 <5.103868,8.977372,5.109262,0.845611,-0.322978,0.128713,0.955290,0.145263>
103050 0.148483 <5.103868,8.977372,5.109262,0.844982,-0.322978,0.128713,0.953393,0.145263>
109548 0.148468 <5.103868,8.977372,5.109262,0.844982,-0.322978,0.128713,0.953279,0.145263>
123085 0.148453 <5.103868,8.977372,5.109262,0.844875,-0.322978,0.128713,0.952906,0.145263>
126062 0.148433 <5.103868,8.977372,5.109262,0.844875,-0.322978,0.128713,0.953704,0.145223>
156819 0.148416 <5.103868,8.977372,5.109262,0.844875,-0.322978,0.128713,0.953635,0.145223>
165310 0.147752 <8.661976,8.266581,8.675038,0.847727,-0.321125,0.143750,0.961544,0.170072>
166715 0.147435 <8.661976,8.266581,8.674150,0.848017,-0.321125,0.143750,0.960990,0.170072>
166790 0.147388 <8.661552,8.266581,8.674150,0.848017,-0.321125,0.143750,0.962439,0.170072>
166810 0.147324 <8.661552,8.266581,8.674150,0.848017,-0.321125,0.143750,0.960948,0.170072>
166850 0.147287 <8.661552,8.266581,8.674150,0.848017,-0.321125,0.143750,0.961705,0.170072>
167003 0.147175 <8.660296,8.266581,8.674150,0.848017,-0.321125,0.143750,0.961574,0.170072>
168291 0.147153 <8.660296,8.266581,8.674150,0.847675,-0.321125,0.143750,0.960014,0.170086>
168447 0.147124 <8.660296,8.266581,8.674150,0.847675,-0.321125,0.143750,0.960369,0.170072>
168734 0.147081 <8.660296,8.266581,8.674150,0.847268,-0.321125,0.143750,0.958502,0.170072>
168792 0.147014 <8.660296,8.266581,8.674150,0.847268,-0.321125,0.143821,0.960359,0.170072>
181032 0.147004 <8.660296,8.266581,8.674150,0.847268,-0.321125,0.143821,0.960197,0.170072>
207341 0.146947 <8.661534,8.266581,8.672929,0.847268,-0.321125,0.143821,0.959081,0.170072>
272288 0.146925 <8.661534,8.266581,8.672929,0.847092,-0.321125,0.143821,0.958466,0.170072>
285515 0.146904 <8.661534,8.266581,8.672929,0.846888,-0.321125,0.143821,0.957996,0.170072>
365170 0.146881 <8.662052,8.266581,8.672929,0.846888,-0.321125,0.143821,0.958096,0.170057>
418452 0.146866 <8.661952,8.266581,8.672929,0.846840,-0.321125,0.143821,0.958008,0.170057>
478173 0.145592 <8.661328,8.266581,8.672929,0.847286,-0.323168,0.143821,0.945135,0.170057>
478349 0.145460 <8.661328,8.266581,8.672929,0.847286,-0.323168,0.143821,0.945654,0.170057>
478493 0.145110 <8.661310,8.266581,8.672929,0.846980,-0.323168,0.143821,0.946462,0.170057>
479176 0.141070 <8.663704,8.266581,8.675541,0.847986,-0.323168,0.143216,0.940551,0.170057>
479247 0.139870 <8.663704,8.266581,8.675541,0.847986,-0.323168,0.143216,0.938316,0.170057>
479254 0.139688 <8.662896,8.266581,8.675541,0.847986,-0.323168,0.143216,0.936226,0.170057>
479406 0.139605 <8.663704,8.266581,8.675541,0.849304,-0.323168,0.143216,0.942002,0.170057>
479450 0.139536 <8.662896,8.266581,8.675541,0.847986,-0.323168,0.143216,0.937165,0.170057>
479597 0.139459 <8.662298,8.266581,8.675541,0.847986,-0.323168,0.143216,0.937165,0.170057>
479749 0.139438 <8.662896,8.266581,8.675541,0.848307,-0.323168,0.143216,0.938612,0.170057>
482941 0.139231 <8.661727,8.266581,8.676446,0.849234,-0.323168,0.143216,0.943283,0.170057>
483012 0.139086 <8.661727,8.266581,8.676446,0.848192,-0.323168,0.143216,0.938335,0.170057>
483026 0.138948 <8.661727,8.266581,8.676446,0.849234,-0.323168,0.143216,0.942328,0.170057>
483043 0.138932 <8.661727,8.266581,8.676446,0.849234,-0.323168,0.143216,0.942132,0.170057>

Projection param: <8.662,8.267,8.676,0.849,-0.323,0.143,0.942,0.170>
```

```
Input data:

input #0 (m): <48.000,0.000,51.000> (px): <618.000,361.000>
 (screen->real): <48.006,0.000,50.996> (error): 0.007018m

input #1 (m): <12.000,0.000,28.000> (px): <375.000,565.000>
 (screen->real): <12.419,0.000,27.869> (error): 0.439199m

input #2 (m): <33.000,0.000,11.000> (px): <974.000,536000000.000>
 (screen->real): <32.997,0.000,11.014> (error): 0.014749m

input #3 (m): <22.000,0.000,62.000> (px): <307.000,387.000>
 (screen->real): <21.908,0.000,62.022> (error): 0.094740m

Average error: 0.138927m
Max error: 0.439199m

Test data:

input #0 (m): <18.000,0.000,25.000> (px): <531.000,543.000>
 (screen->real): <18.882,0.000,25.527> (error): 1.026980m

input #1 (m): <35.000,0.000,30.000> (px): <692.000,436.000>
 (screen->real): <36.008,0.000,31.244> (error): 1.601299m

Average error: 1.314115m
Max error: 1.601299m
```



param.txt:

{"_dim":"8","_val":["8.661727","8.266581","8.676446","0.849234","-0.323168","0.143216","0.941986","0.170057"]}