```
(*Reconstruct Scene with Sampson
 Approximation_____________________________________*)
StructureComputation[F_, P3_, PC1_, PC2_, K1_, K2_, RForOk2_, tForOK2_] :=
  Module[{T, TPrime, StructureF , R,  RPrime, RotatedF,  f ,  fPrime,  a, b,
  Modul
    c, d, rootMin, l, lPrime, K2Ex, K1Ex, P, PPrime, CoefficientMtx = {},
    ee, ii, SpacePoints = {}, SpacePointsMMNotScaled = {}},

   For[vv = 1, vv ≤  Length[PC1mm], vv++,
   For-Schleife          Länge
    K2Ex = P3;

    (*Translate Points to origin*)
      verschiebe
    T = {{1 , 0, -PC1[[vv, 1]]}, {0, 1, -PC1[[vv, 2]]}, {0, 0, 1}};
    TPrime = {{1 , 0, -PC2[[vv, 1]]}, {0, 1, -PC2[[vv, 2]]}, {0, 0, 1}};

    StructureF  =  Transpose[Inverse[TPrime]].F.Inverse[T];
                   transponiere  inverse Matrix          inverse Matrix
    (*Normalize epipoles*)
      normalisiere
    epipole = Flatten[NullSpace[StructureF]];
              ebne ein  Nullraum
    epipolePrime  =  Flatten[NullSpace[Transpose[StructureF]]];
                     ebne ein  Nullraum  transponiere

    Teste  =  epipole[[1]]^2 + epipole[[2]]^2;
    TestePrime  =  epipolePrime[[1]]^2 + epipolePrime[[2]]^2;

    ScaleE = 1 / Teste;
    ScaleEPrime = 1 / TestePrime;

    epipole =
     {epipole[[1]]^2 * ScaleE, epipole[[2]]^2 * ScaleE, epipole[[3]]^2 * ScaleE};
    epipole = {Sqrt[epipole[[1]]], Sqrt[epipole[[2]]], Sqrt[epipole[[3]]]};
              Quadratwurzel          Quadratwurzel          Quadratwurzel
    epipolePrime = {epipolePrime[[1]]^2 * ScaleEPrime,
      epipolePrime[[2]]^2 * ScaleEPrime, epipolePrime[[3]]^2 * ScaleEPrime};
    epipolePrime = {Sqrt[epipolePrime[[1]]], Sqrt[epipolePrime[[2]]],
                    Quadratwurzel             Quadratwurzel
      Sqrt[epipolePrime[[3]]]};
      Quadratwurzel

    Teste  =  epipole[[1]]^2 + epipole[[2]]^2;
    TestePrime  =  epipolePrime[[1]]^2 + epipolePrime[[2]]^2;

    (*Rotate Epipoles to (1,0,0^T)*)
      drehe
    R =
     {{epipole[[1]], epipole[[2]], 0}, {-epipole[[2]], epipole[[1]], 0}, {0, 0, 1}};
    RPrime  =  {{epipolePrime[[1]], epipolePrime[[2]], 0},
      {-epipolePrime[[2]], epipolePrime[[1]], 0}, {0, 0, 1}};

    RTest =  Transpose[R].R;
             transponiere
```

```
RPrimeTest = Transpose[RPrime].RPrime;
Rep = R.epipole;
RePrime = RPrime.epipolePrime;

(*Replace F with RPrime.StructureF.R^T*)

RotatedF = RPrime.StructureF.Transpose[R];

(*Set variables f, fPrime, a, b, c, d for further computations*)

f = epipole[[3]];
fPrime = epipolePrime[[3]];
a = RotatedF[[2, 2]];
b = RotatedF[[2, 3]];
c = RotatedF[[3, 2]];
d = RotatedF[[3, 3]];

(*Form the polynomial g(t)*)

roots = Solve[root ((a * root + b) ^2 + fPrime^2 (c * root + d) ^2) ^2 -
    (a * d - b * c) * (1 + f^2 * root^2) ^2 * (a * root + b) * (c * root + d) == 0, root];

rootsReals =
 Solve[root ((a * root + b) ^2 + fPrime^2 (c * root + d) ^2) ^2 - (a * d - b * c) *
     (1 + f^2 * root^2) ^2 * (a * root + b) * (c * root + d) == 0, root, Reals];

tInfinity = 1 / f^2 + c^2 / (a^2 + fPrime^2 * c^2);

(*evaluate min cost function of real root values in s(t)*)
st = {};

For[oo = 1, oo ≤ Length[rootsReals], oo++,
 AppendTo[st, (root^2 / (1 + f^2 * root^2)) + (c * root + d) ^2 /
      ((a * root + b) ^2 + fPrime^2 * (c * root + d) ^2) /. rootsReals[[oo]]];

];


rootMin = Min[st];
l = {rootMin * f, 1, -rootMin};

lPrime = {-fPrime * (c * rootMin + d), a * rootMin + b, c * rootMin + d};
```

```
closestPointX = {-l[[1]] * l[[3]], -l[[2]] * l[[3]], l[[1]]^2 + l[[2]]^2};
closestPointX = Inverse[T].Transpose[R].closestPointX;
                inverse Matrix  transponiere
closestPointX = closestPointX / closestPointX[[3]];

closestPointXPrime = {-lPrime[[1]] * lPrime[[3]],
  -lPrime[[2]] * lPrime[[3]], lPrime[[1]]^2 + lPrime[[2]]^2};
closestPointXPrime = Inverse[TPrime].Transpose[RPrime].closestPointXPrime;
                     inverse Matrix          transponiere
closestPointXPrime = closestPointXPrime / closestPointXPrime[[3]];

Print["closestPointX = ", closestPointX];
gib aus
Print["closestPointXPrime= ", closestPointXPrime];
gib aus

K1Ex = {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}};

P = K1.K1Ex;
PPrime = K2.K2Ex;


(* Reconstruct Points with linear Triangulation Method*)
                                                  Methode

A = {closestPointX[[1]] * P[[3, All]] - P[[1, All]],
                                  alle        alle
  closestPointX[[2]] * P[[3, All]] - P[[2, All]],
                               alle        alle
  closestPointXPrime[[1]] * PPrime[[3, All]] - PPrime[[1, All]],
                                        alle              alle
  closestPointXPrime[[2]] * PPrime[[3, All]] - PPrime[[2, All]]};
                                        alle              alle

SVDTest = SingularValueDecomposition[A];
          Singulärwertzerlegung
SpacePointPx =
 {SVDTest[[3, 1, 4]], SVDTest[[3, 2, 4]], SVDTest[[3, 3, 4]], SVDTest[[3, 4, 4]]};

SpacePointPx = SpacePointPx / SpacePointPx[[4]];

AppendTo[SpacePointsMMNotScaled, SpacePointPx];
hänge an bei
SpacePointsMMNotScaled = Map[# / #[[4]] &, SpacePointsMMNotScaled];
                         wende an

Clear[st];
lösche
];
GraficPoints = Map[{#[[1]], #[[2]], #[[3]]} &, SpacePointsMMNotScaled];
               wende an
GraficPoints2 = Map[{#[[1]], #[[2]], #[[3]]} &, SpacePointsMMNotScaled];
                wende an
plrange = {-1, 1};
```

```
Print[ListPointPlot3D[GraficPoints, AxesLabel → {x, y, z}, BoxRatios → 1,
  PlotRange → (*{plrange,plrange,plrange}*)All, PlotStyle → PointSize[0.023]]];

Print["RForOk2 = ", RForOk2];
Print["tForOK2 = ", tForOK2];
t = RForOk2.tForOK2;
t = {t[[1]], t[[2]], t[[3]]};
Print["t = ", t];

Print["Reconstructed Points 3D = ", GraficPoints];
G1 = Graphics3D[{Red, PointSize[0.02], Point[{0, 0, 0}]}];
G2 = Graphics3D[{Blue, PointSize[0.01], Point[GraficPoints]}];
G3 = Graphics3D[{Green, PointSize[0.02], Point[t]}];
Print[Show[G1, G3, G2, Axes → True,
  PlotRange → All, AxesLabel → {x, y, z}(*,BoxRatios→1*)]];

TwoDGraphicPoints = SpacePointsMMNotScaled;

TwoDGraphicPoints = Map[{#[[1]] / #[[3]], #[[2]] / #[[3]]} &, TwoDGraphicPoints];
TwoDGraphicPoints = Map[{#[[1]], #[[2]]} &, TwoDGraphicPoints];
Print[ListPlot[TwoDGraphicPoints,
  AxesLabel → {x, y}, PlotRange → All, PlotStyle → PointSize[0.01]]];

];
```