Slide 1

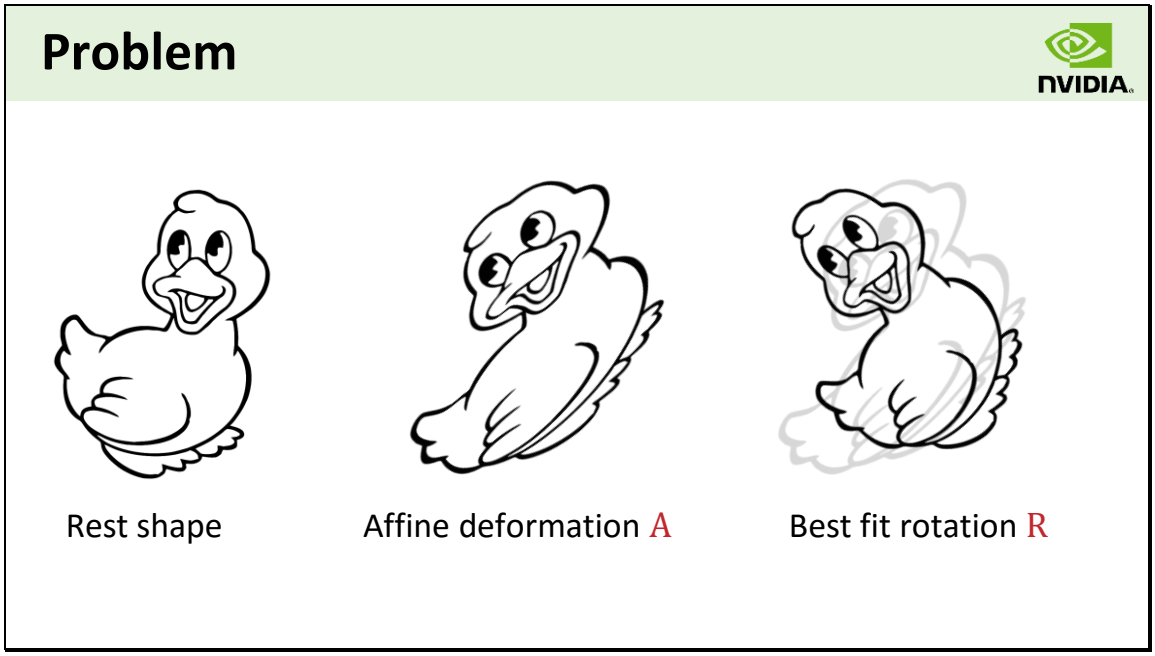
A Robust Method to Extract the Rotational Part of Deformations

Matthias Müller[1] Jan Bender[2] Nuttapong Chentanez[1] Miles Macklin[1]

1 NVIDIA

2 RWTH AACHEN UNIVERSITY

Slide 2



Here is the problem we are addressing with our new method.
Given a rest shape of an object
and an affine transformation of it described by a matrix A
We want to find a rotation matrix R that best fits the affine transformation

Slide 3

Two Main Applications

- Co-rotational FEM from 4 tetrahedral nodes

$$\mathrm{A} = [\mathbf{p}_2 - \mathbf{p}_1, \mathbf{p}_3 - \mathbf{p}_1, \mathbf{p}_4 - \mathbf{p}_1\,] \\ [\mathbf{q}_2 - \mathbf{q}_1, \mathbf{q}_3 - \mathbf{q}_1, \mathbf{q}_4 - \mathbf{q}_1\,]^{-1}$$

- Shape Matching from point cloud

$$\mathrm{A} = \left(\sum_i m_i\, \mathbf{p}_\mathrm{i} \mathbf{q}_\mathrm{i}{}^\mathrm{T}\right)\left(\sum_i m_i\, \mathbf{q}_\mathrm{i} \mathbf{q}_\mathrm{i}{}^\mathrm{T}\right)^{-1}$$

There are two main applications in graphics in which this problem arises
The first is co-rotational FEM
The general deformation of a tetrahedron with vertices p1, p2, p3, and p4 can be described by an affine transformation
To extract the rotational part of this deformation, we need the closest rotation
To compute the optimal rigid transform in shape matching, we first compute the matrix A like this and then again have to find the closest R

Slide 4

Standard Approach

- Use Polar Decomposition

  $A = RS$ where R is a rotation matrix and S is symmetric

- We have

  $A^T A = S^T R^T R S = S^T S = S^2$

- So we can compute

  $S = \sqrt{A^T A}$ and $R = AS^{-1}$

The standard approach to find the closest matrix R to A is to compute the polar decomposition splitting A into a rotation matrix R and a symmetric matrix S
This can be done as follows
Let us have a look at the expression A transposed A. If we substitute RS for A we get S transposed R transposed R S.
Since R is a rotation matrix R transposed R is the identity.
And since S is symmetric S transposed S is S squared
So we can directly compute S as the square root of A transposed A
Once we know S we compute R as A times the inverse of S

Slide 5

Standard Approach

- To compute $S^{-1} = \sqrt{A^T A}^{-1}$
- Since $A^T A$ is symmetric we can decompose it as $A^T A = UDU^T$ with U a rotation and D diagonal (Jacobi iterations)
- This makes computing the inverse square root easy:
  $\sqrt{A^T A}^{-1} = \sqrt{UDU^T}^{-1} = U\sqrt{D}^{-1}U^T$
- Compute the inverse square roots of the scalar diagonal elements (singular values of A)

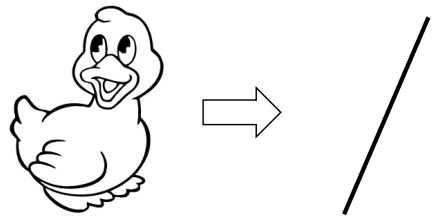
So we have to compute the inverse of S and the inverse square root of A transposed A
Since S is symmetric we can decompose it into its eigenvectors in a matrix U and the real eigenvalues on the diagonal matrix D
This makes computing the inverse square root easy. We simply have to take the inverse square roots of the eigenvalues, which are the scalar values on the diagonal of D.

Slide 6

Difficulties

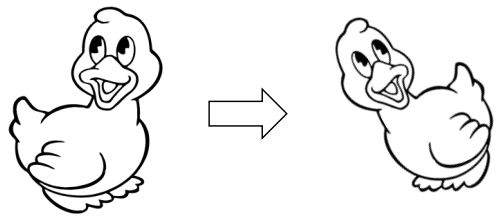
- Degenerated cases
  co-planar, co-linear points

  $\det(A) = 0$



- Inverted configuration

  $\det(A) < 0$

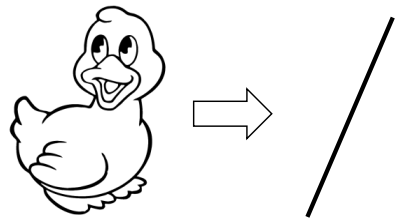


There are two main difficulties with this approach.
For degenerated cases when points or elements become close to co-planar or co-linear, the determinant of A becomes close to zero
In this case computing the inverse becomes instable or even impossible.
Intuitively, the optimal rotation in such cases is not unique
The second difficult case is when A is inverting. This is the case when its determinate is smaller than one.
In this case the polar decomposition yields an inverting R instead of a proper rotation

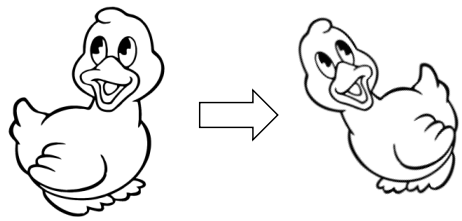
Slide 7

Solutions

- Choose missing eigenvectors
  - One missing: Cross product
  - Two or three missing?



- Choose flip axis
  - Smallest eigenvalue [Irving et al. 04] Standard approach
  - Smallest extent of element [Schmedding et al. 08]



A solution to the first problem is to somehow compute the missing eigenvectors based on the existing ones
If only one eigenvector is missing, it can be computed as the cross product from the other two
If two are missing, it is not clear what to do. Once could compute a cross product with a canonical axis for instance
If all three are missing than the identity might be a choice.
Those cases typically yield temporal incoherence

The common solution to the inversion problem is the one suggested by Irving et al.
They simply flip the direction of the eigenvector corresponding to the smallest eigenvalue.
Schmedding et al found that artifacts can be reduced by choosing the eigenvector along which the element has the smallest extent.

Slide 8

Putting it all together [Irving et al. 04]

```
void jacobiRotate(Matrix3d &A, Matrix3d &R, int p, int q)
{
   if (A(p, q) == 0.0)
      return;

   double d = (A(p, p) - A(q, q)) / (2.0*A(p, q));
   double t = 1.0 / (fabs(d) + sqrt(d*d + 1.0));
   if (d < 0.0)
      t = -t;
   double c = 1.0 / sqrt(t*t + 1);
   double s = t*c;
   A(p, p) += t*A(p, q);
   A(q, q) -= t*A(p, q);
   A(p, q) = A(q, p) = 0.0;

   int k;
   for (k = 0; k < 3; k++)
   {
      if (k != p && k != q)
      {
         double Akp = c*A(k, p) + s*A(k, q);
         double Akq = -s*A(k, p) + c*A(k, q);
         A(k, p) = A(p, k) = Akp;
         A(k, q) = A(q, k) = Akq;
      }
   }
   for (k = 0; k < 3; k++)
   {
      double Rkp = c*R(k, p) + s*R(k, q);
      double Rkq = -s*R(k, p) + c*R(k, q);
      R(k, p) = Rkp;
      R(k, q) = Rkq;
   }
}

void eigenDecomposition(const Matrix3d &A, Matrix3d &
      eigenVecs, Vector3d &eigenVals)
{
   const int numJacobiIterations = 10;
   const double epsilon = 1e-15;

   Matrix3d D = A;

   eigenVecs.setIdentity();
   int iter = 0;
   while (iter < numJacobiIterations)
   {
      int p, q;
      double a, max;
      max = fabs(D(0, 1));
      p = 0;
      q = 1;
      a = fabs(D(0, 2));
      if (a > max)
      {
         p = 0;
         q = 2;
         max = a;
      }
      a = fabs(D(1, 2));
      if (a > max)
      {
         p = 1;
         q = 2;
         max = a;
      }
      if (max < epsilon)
         break;
      jacobiRotate(D, eigenVecs, p, q);
      iter++;
   }
   eigenVals[0] = D(0, 0);
   eigenVals[1] = D(1, 1);
   eigenVals[2] = D(2, 2);
}
```

```
void rotationMatrixIrving(const Matrix3d &A, Matrix3d
{
   Matrix3d AT_A, V;
   AT_A = A.transpose() * A;

   Vector3d S;
   eigenDecomposition(AT_A, V, S);

   const double detV = V.determinant();
   if (detV < 0.0)
   {
      double minLambda = DBL_MAX;
      unsigned char pos = 0;
      for (unsigned char l = 0; l < 3; l++)
      {
         if (S[l] < minLambda)
         {
            pos = l;
            minLambda = S[l];
         }
      }
      V(0, pos) = -V(0, pos);
      V(1, pos) = -V(1, pos);
      V(2, pos) = -V(2, pos);
   }

   if (S[0] < 0.0f)
      S[0] = 0.0f;
   if (S[1] < 0.0f)
      S[1] = 0.0f;
   if (S[2] < 0.0f)
      S[2] = 0.0f;

   Vector3d sigma;
   sigma[0] = sqrt(S[0]);
   sigma[1] = sqrt(S[1]);
   sigma[2] = sqrt(S[2]);

   unsigned char chk = 0;
   unsigned char pos = 0;
   Matrix3d U;
   for (unsigned char l = 0; l < 3; l++)
   {
      if (fabs(sigma[l]) < 1.0e-4)
      {
         pos = l;
         chk++;
      }
   }

   if (chk > 0)
   {
      if (chk > 1)
      {
         U.setIdentity();
      }
      else
      {
         U = A * V;
         for (unsigned char l = 0; l < 3; l++)
         {
            if (l != pos)
            {
               for (unsigned char m = 0; m < 3; m++)
               {
                  U(m, l) += 1.0f / sigma[l];
               }
            }
         }

         Vector3d v[2];
         unsigned char index = 0;
         for (unsigned char l = 0; l < 3; l++)
         {
            if (l != pos)
            {
               v[index++] = Vector3d(U(0, l), U(1, l), U
                     (2, l));
            }
         }
         Vector3d vec = v[0].cross(v[1]);
         vec.normalize();
         U(0, pos) = vec[0];
         U(1, pos) = vec[1];
         U(2, pos) = vec[2];
      }
   }
   else
   {
      Vector3d sigmaInv=(1.0 / sigma[0], 1.0 / sigma[1],
            1.0 / sigma[2]);
      U = A * V;
      for (unsigned char l = 0; l < 3; l++)
      {
         for (unsigned char m = 0; m < 3; m++)
         {
            U(m, l) += sigmaInv[l];
         }
      }
   }

   const double detU = U.determinant();

   if (detU < 0.0)
   {
      double minLambda = DBL_MAX;
      unsigned char pos = 0;
      for (unsigned char l = 0; l < 3; l++)
      {
         if (sigma[l] < minLambda)
         {
            pos = l;
            minLambda = sigma[l];
         }
      }

      sigma[pos] = -sigma[pos];
      U(0, pos) = -U(0, pos);
      U(1, pos) = -U(1, pos);
      U(2, pos) = -U(2, pos);
   }

   R = U * V.transpose();
}
```

Putting all this together yields the following algorithm. For convenience, we have it in the appendix of the paper
As you can see, it is quite long and has many branches

Slide 9

Motivation for a New Approach

- Make algorithm simpler (better fit on GPU)
- Remove need of choices

- Basic idea:

  Input is A **and** R_{prev}

- Previous R is used to fill in missing information
- Natural for simulations (temporal coherence)

So the motivation for us was to find a simpler algorithm that better fits on the GPU
Also we wanted to remove the need of choosing missing eigenvectors or flipping axes
We solve these problems with a simple idea
Instead of taking only A as input, we take A and the R from the previous time step or from a previous iteration
When can then use R to fill in the missing information
This is a natural solution for simulation since it guarantees temporal coherence.

Slide 10

The Algorithm

- Our algorithm has the form

$$\boxed{\mathrm{R} \leftarrow exp(\omega)\ \mathrm{R}_{\mathrm{prev}}}$$

  where $exp(\omega)$ is the rotation matrix with axis $\omega/|\omega|$ and angle $|\omega|$.

- Notice that $\det\big(exp(\omega)\big) = +1$ always holds even for $\omega = 0$ since $exp(0) = \mathrm{I}$

- If $\mathrm{R}_{\mathrm{prev}}$ is a proper rotation matrix then so is R

- How to find ω?

We choose our algorithm to have the following form
We compute the new R by rotating the previous R.
The exponential map exp omega is the rotation matrix along the direction of omega about the angle omega
Notice that the exponential map is always a proper rotation matrix with determinant plus one even if omega is zero.
In that case we get the identity.
So if the previous R was a proper rotation matrix then so is R
The remaining question is now how to find omega

Slide 11

Physical Interpretation

- The R of the polar decomposition minimizes the Frobenius norm

  $$\mathrm{F}(\mathrm{R}) = \|\mathrm{A} - \mathrm{R}\|^2 = \Sigma(\mathbf{r}_\mathrm{i} - \mathbf{a}_\mathrm{i})^2$$

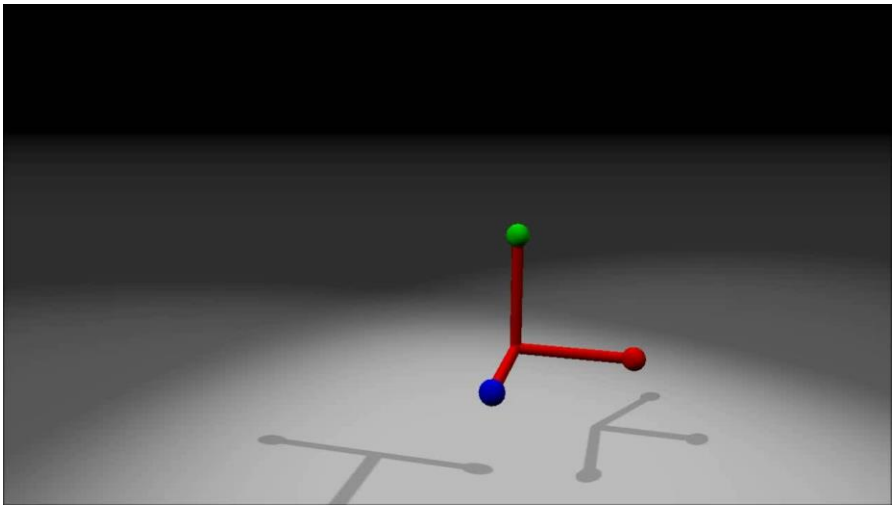
  i.e. is the rotation matrix that is closest to A

- Idea: interpret
  - The column vectors $\mathbf{a}_\mathrm{i}$ of A as a static object
  - The column vectors $\mathbf{r}_\mathrm{i}$ of R as a dynamic **rigid** object
  - $\mathrm{F}(\mathrm{R})$ as an energy

To find omega we use a physical interpretation of the problem
As we show in the paper, the rotation matrix R of the polar decomposition minimizes the Frobenius norm F which measures the Euclidean distance between A and R. In other words, the R of the polar decomposition is the rotation matrix that is closest to A in this norm
The frombenius norm can also be expressed terms of the column vectors of the two matrices.
This yields the following physical interpretation of the problem
We treat the column vectors ai of A as a static object
We treat the column vectors ri or R as a dynamic but rigid object
And we treat F of R as an energy to be minimized

Slide 12

Demo



Here you see a simulation of this setup
First, the user manipulates A.
During this, the forces due to the given energy act on the dynamic but rigid object R.
As you can see, R constantly tries to stay as close as possible to A
We can also keep A constant and manipulate R with the same effect
By making certain axes of A very small we decrease the condition number of A and make the problem close to singular
Our method still yields plausible results in this case

Slide 13

Torque

- The energy $E(R) = \frac{1}{2}\sum(\mathbf{r}_i - \mathbf{a}_i)^2$

  yields the forces $\mathbf{f}_i = \mathbf{a}_i - \mathbf{r}_i$

  on the axes and a total toque of

$$\tau = \sum \mathbf{r}_i \times (\mathbf{a}_i - \mathbf{r}_i) = \sum \mathbf{r}_i \times \mathbf{a}_i$$

So we choose ω to be parallel to τ.

But thow to choose the angle $|\omega|$?

Mathematically we first define the energy in terms of the axes of R and A
This energy yields the simple forces fi = ai – ri which act on the axes of R
The forces result in a torque tau acting on the rigid object R which turns out to be the sum of the cross products of the axes of R and A
Since the forces start to spin R along the direction of tau we choose omega to be parallel to tau.
The remaining question is how to choose the magnitude of omega

Slide 14

Angle

- For a single pair of axes $\mathbf{r}_i$ and $\mathbf{a}_i$
  we want $|\omega|$ to be their mutual angle.

- If we choose $\omega = \dfrac{\mathbf{a}_i \times \mathbf{r}_i}{\mathbf{a}_i \cdot \mathbf{r}_i}$

  we get $|\omega| = \dfrac{|\mathbf{a}_i||\mathbf{r}_i| \sin\phi}{|\mathbf{a}_i||\mathbf{r}_i| \cos\phi} = \tan\phi \approx \phi$ for small ϕ

Let us have a look at a single pair of axes ri and ai
In this case we want omega to be their mutual angle because we want omega to align r with a
If we choose omega to be the cross product divided by the scalar product then we get for the magnitude of omega the tangent of the mutual angle which is the mutual angle for small angles.

Slide 15

The Final Formula

- Our final formula reads:

$$\mathrm{R} \leftarrow exp\left(\frac{\sum_i \mathbf{r}_i \times \mathbf{a}_i}{\left|\sum_i \mathbf{r}_i \cdot \mathbf{a}_i\right| + \varepsilon}\right)\mathrm{R}_{prev}$$

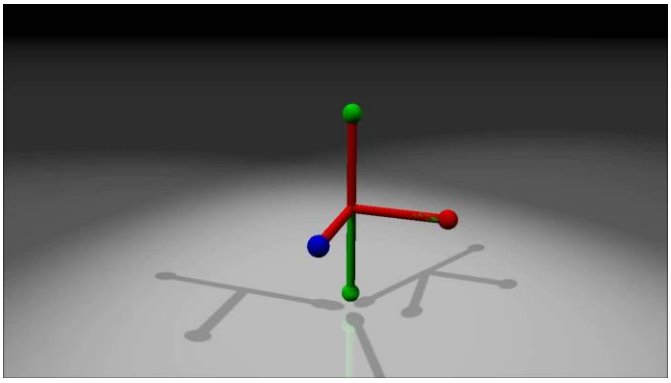
where $\varepsilon = 10^{-9}$ is a safety parameter

This yields are final very simple formula
Notice that we take the absolute value of the denominator to not change the sign of the torque.
We also added an epsilon to make the division safe.

Slide 16

Energy Maxima

- Torque is zero as well when $E(R)$ is maximized
- Zero set
- no problem in practice
- resolved by small perturbation



There is one property of the physical interpretation that needs to be noticed
The torque is not only zero at the energy minimum but also at energy maxima
Fortunately, the set of configurations for which the energy is maximized is a zero set
It is therefore very unlikely to end up in such a configuration in practice
We haven't seen any artifacts in our simulations
Also, such situations are resolved by a very small perturbation as this demo shows

Slide 17

Source Code

```
void extractRotation(const Matrix3d &A, Quaterniond &q,
              const unsigned int maxIter)
{
  for (unsigned int iter = 0; iter < maxIter; iter++)
  {
    Matrix3d R = q.matrix();
    Vector3d omega =(R.col(0).cross(A.col(0)) + R.col (1).cross(A.col(1)) +
            R.col(2).cross(A.col(2)) ) * (1.0 / fabs(R.col(0).dot(A.col(0)) + R.col
            (1).dot(A.col(1)) + R.col(2).dot(A.col(2))) + 1.0e-9);
    double w = omega.norm();
    if (w < 1.0e-9)
      break;
    q = Quaterniond(AngleAxisd(w, (1.0 / w) * omega)) * q;
    q.normalize();
  }
}
```

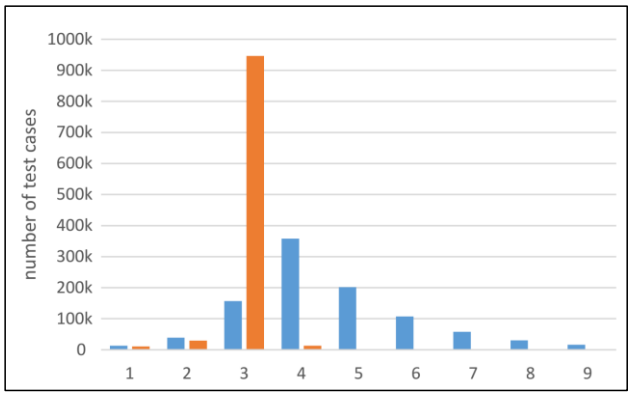
- Much simpler than Irving, no branches
- Up to 2 times faster (depends on iteration count)

The resulting source code using the library Eigen and quaternions to represent R looks like this
As you can see, it is significantly simpler than Irving and has no branches
Dependent on the iteration count it is up to two times faster too

Slide 18

Results

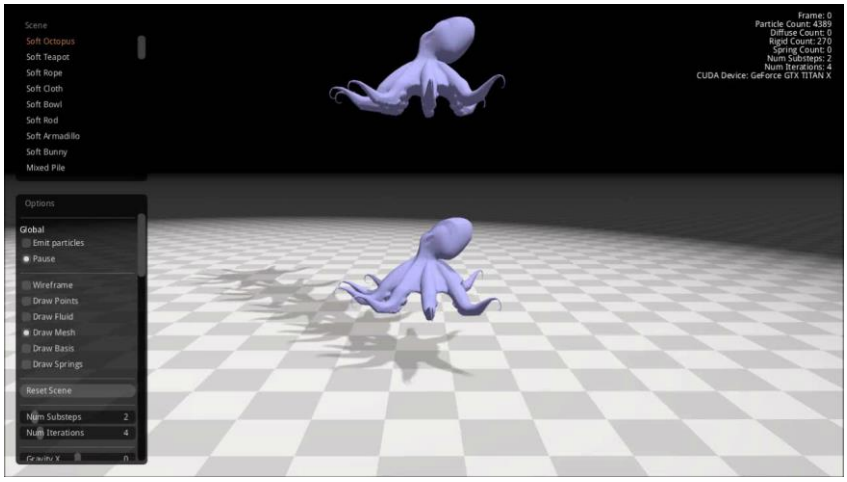
- Blue bars (cold start)
  - R random rotation, $A=I$
  - # iters until $F(R)<0.001$
- Orange bars (warm start)
  - Euler angles of R in $\left[-\frac{\pi}{3},\frac{\pi}{3}\right]$



We did some tests on the convergence of our method
In our setup we set A to be the identity matrix and R to be a random rotation matrix
The blue bars shows the distribution of the number of iterations it took to reduce the Frobenius norm to below 0.001
The orange bars show the effect of a warm start
Here we reduced the Euler angles of R to be within –pi/3 and pi/3
Now 3 iterations are sufficient in all cases. Therefore we set the number of solver iterations to 3 in our examples

Slide 19

In Action



Questions?

Here you see a couple of scenes in which we used our method in the context of shape matching
Are there questions?