A Flexible Tool for Shape Analysis

IDP Presentation

Emanuel Laude
Zorah Lähner

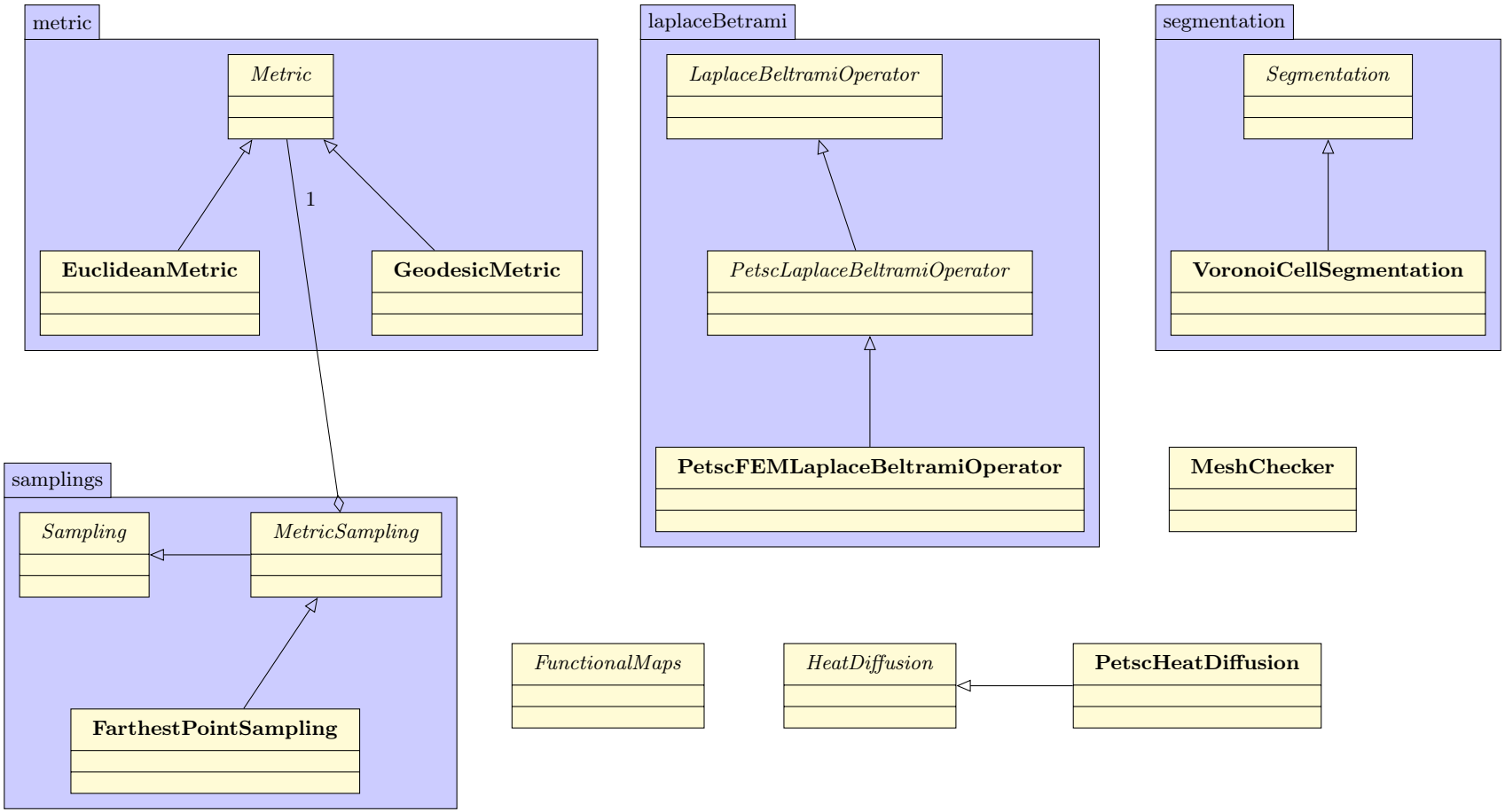
Technische Universität München

January 16, 2015

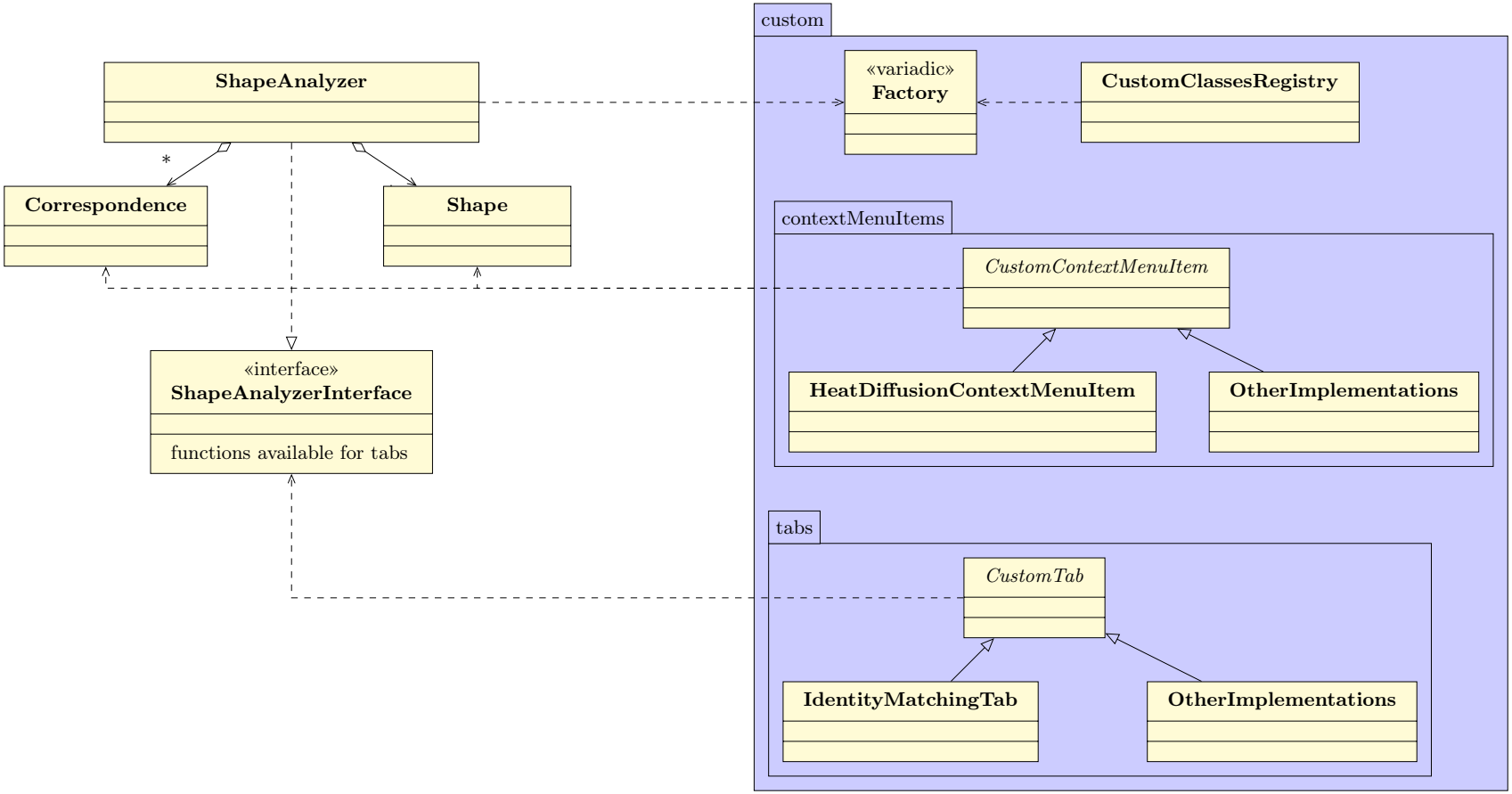
Outline

Demo Time

Class Diagram: Domain



Class Diagram: View



CustomContextMenuItem

or

CustomTab?

Example Menu Item


```
template<class T = Metric>
class VoronoiCellsContextMenuItem : public CustomContextMenuItem
    {
public:
VoronoiCellsContextMenuItem<T>
(shared_ptr<Shape> shape, ShapeAnalyzerInterface* shapeAnalyzer)
 :  CustomContextMenuItem(shape, shapeAnalyzer) { }

   virtual void onClick(vtkIdType pointId, vtkIdType faceId,
       QWidget* parent) {
       (...)  }
};
```

Example Menu Item


```
template<class T = Metric>
class VoronoiCellsContextMenuItem :  public CustomContextMenuItem
    {
public:
VoronoiCellsContextMenuItem<T>
(shared_ptr<Shape> shape, ShapeAnalyzerInterface* shapeAnalyzer)
 :  CustomContextMenuItem(shape, shapeAnalyzer) { }

   virtual void onClick(vtkIdType pointId, vtkIdType faceId,
       QWidget* parent) {
       (...)  }
};
```

Example Menu Item


```
template<class T = Metric>
class VoronoiCellsContextMenuItem :  public CustomContextMenuItem
    {
public:
VoronoiCellsContextMenuItem<T>
 (shared_ptr<Shape> shape, ShapeAnalyzerInterface* shapeAnalyzer)
  : CustomContextMenuItem(shape, shapeAnalyzer) { }

    virtual void onClick(vtkIdType pointId, vtkIdType faceId,
        QWidget* parent) {
       (...)  }
};
```

Example Menu Item


```
template<class T = Metric>
class VoronoiCellsContextMenuItem :  public CustomContextMenuItem
    {
public:
VoronoiCellsContextMenuItem<T>
(shared_ptr<Shape> shape, ShapeAnalyzerInterface* shapeAnalyzer)
 :  CustomContextMenuItem(shape, shapeAnalyzer) { }

    virtual void onClick(vtkIdType pointId, vtkIdType faceId,
        QWidget* parent) {
        (...) }
};
```

Example Menu Item


```
virtual void onClick(...)  {
    bool ok;
    vtkIdType source = QInputDialog::getInt(...);
    if (!ok) { return; }
    (...)
    if(ok) {
        try {
            auto m = make_shared<T>(shape_);
            auto fps = make_shared<FarthestPointSampling>(shape_,
                m, source, numberOfSegments);
            VoronoiCellSegmentation segmentation(shape_, m, fps);
            shape_->setColoring(segmentation.getSegments(),
             Shape::Coloring::Type::PointSegmentation);
        } catch(metric::MetricError& e) {
            QMessageBox::warning(parent, "Exception", e.what());
        }}}
```

Example Menu Item

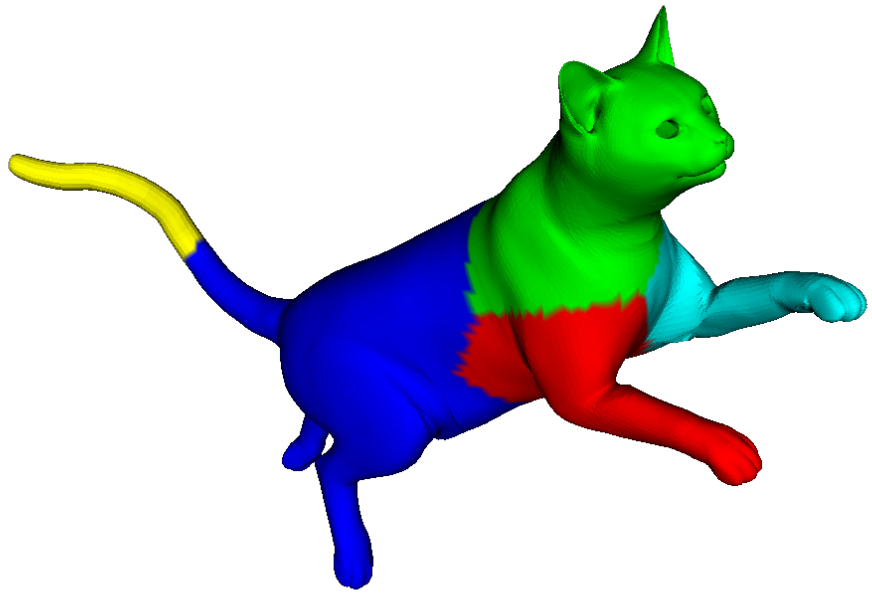

```
virtual void onClick(...)  {
    bool ok;
    vtkIdType source = QInputDialog::getInt(...);
    if (!ok) { return; }
    (...)
    if(ok) {
        try {
            auto m = make_shared<T>(shape_);
            auto fps = make_shared<FarthestPointSampling>(shape_,
                m, source, numberOfSegments);
            VoronoiCellSegmentation segmentation(shape\_, m, fps);
            shape_->setColoring(segmentation.getSegments(),
             Shape::Coloring::Type::PointSegmentation);
        } catch(metric::MetricError& e) {
            QMessageBox::warning(parent, "Exception", e.what());
        }}}
```

Example Menu Item


```
virtual void onClick(...)  {
    bool ok;
    vtkIdType source = QInputDialog::getInt(...);
    if (!ok) { return; }
    (...)
    if(ok) {
        try {
            auto m = make_shared<T>(shape_);
            auto fps = make_shared<FarthestPointSampling>(shape_,
                m, source, numberOfSegments);
            VoronoiCellSegmentation segmentation(shape_, m, fps);
            shape_->setColoring(segmentation.getSegments(),
             Shape::Coloring::Type::PointSegmentation);
        } catch(metric::MetricError& e) {
            QMessageBox::warning(parent, "Exception", e.what());
        }}}
```

Shape Coloring

- PointSegmentation
- FaceSegmentation
- PointRGB
- FaceRGB
- PointScalar
- FaceScalar

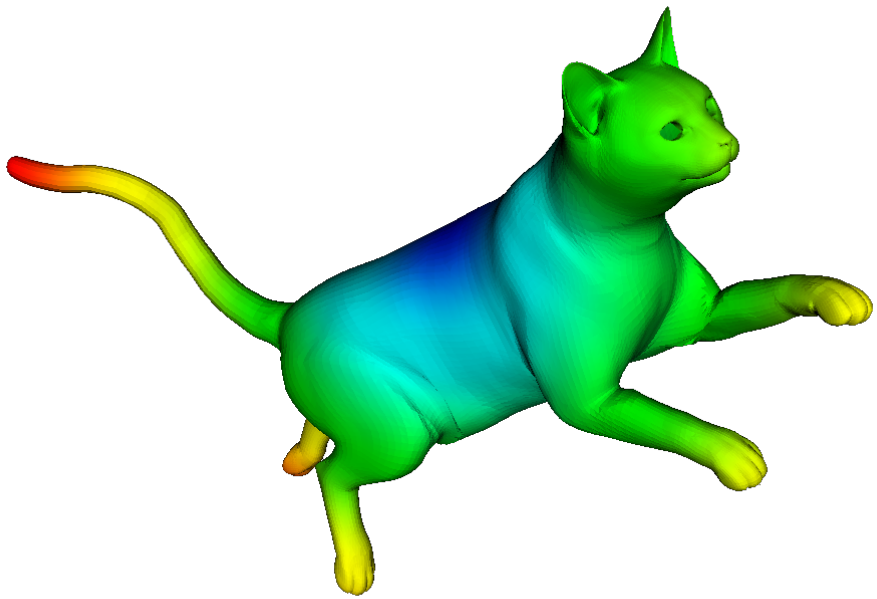


Shape Coloring

- PointSegmentation
- FaceSegmentation
- PointRGB
- FaceRGB
- PointScalar
- FaceScalar

Shape Coloring

- PointSegmentation
- FaceSegmentation
- PointRGB
- FaceRGB
- PointScalar
- FaceScalar



CustomTab

- Constructor signature:
  - `HashMap<vtkActor*, shared_ptr<Shape»& shapes`
  - `HashMap<shared_ptr<PointCorrespondence>, bool>& pointCorrespondences`
  - `HashMap<shared_ptr<FaceCorrespondence>, bool>& faceCorrespondences`
  - `ShapeAnalyzerInterface* shapeAnalyzer`
- Abstract functions:
  - `void onShapeAdd(Shape* shape)`
  - `void onShapeEdit(Shape* shape)`
  - `void onShapeDelete(Shape* shape)`
  - `void onClear()`

Connect Custom Classes


```
template<class T, class... Args>
class Factory { ... }
typedef Factory<CustomContextMenuItem, shared_ptr<Shape>,
    ShapeAnalyzerInterface*> CustomContextMenuItemFactory;

typedef Factory<CustomTab, const HashMap<vtkActor*>,
    shared_ptr<Shape»&,
    const HashMap<shared_ptr<PointCorrespondence>, bool>&,
    const HashMap<shared_ptr<FaceCorrespondence>, bool>&,
    ShapeAnalyzerInterface*> CustomTabFactory;
```

Connect Custom Classes

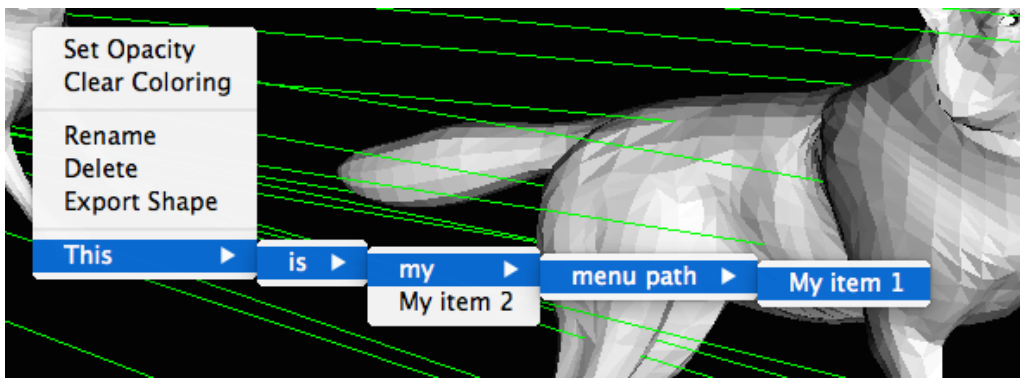

```
template<class T, class...  Args>
class Factory { ...  }

typedef Factory<CustomContextMenuItem, shared_ptr<Shape>,
    ShapeAnalyzerInterface*> CustomContextMenuItemFactory;

typedef Factory<CustomTab, const HashMap<vtkActor*,
    shared_ptr<Shape>>&,
    const HashMap<shared_ptr<PointCorrespondence>, bool>&,
    const HashMap<shared_ptr<FaceCorrespondence>, bool>&,
    ShapeAnalyzerInterface*> CustomTabFactory;
```

Connect Custom Classes


```
struct CustomClassesRegistry {
    static void registerContextMenuItems() {
       CustomContextMenuItemFactory::getInstance()->Register<
           MyMenuItemClass>("key", "My␣Menu␣Item␣Label");
    }
    static void registerTabs() {
        CustomTabFactory::getInstance()->Register<MyTabClass>("
            key", "My␣Tab␣Label");
    }
}
```

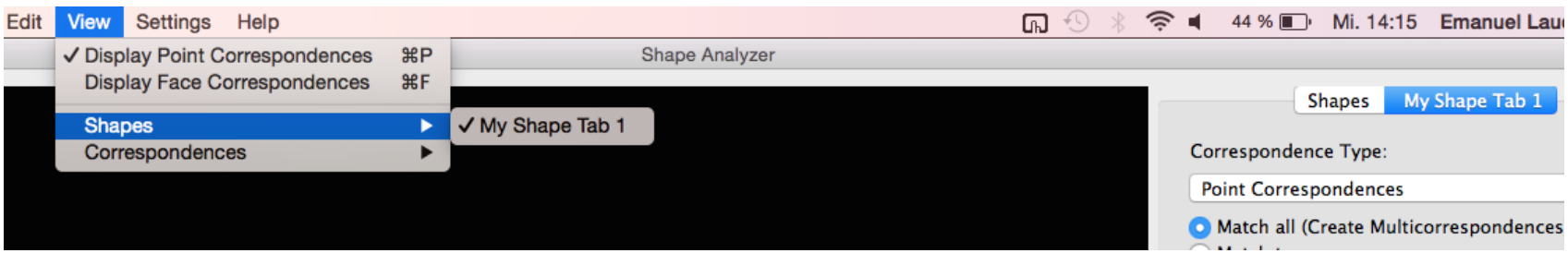
Connect Custom Menu Items


```
static void registerContextMenuItems() {
    CustomContextMenuItemFactory::getInstance()->Register<
        ExtractSegmentContextMenuItem>("extract_segment",
       "This>>is>>my>>menu␣path>>My␣item␣1");

    CustomContextMenuItemFactory::getInstance()->Register<
        VoronoiCellsContextMenuItem<GeodesicMetric>>("
        voronoicells_geodesic",
       "This>>is>>My␣item␣2");
}
```



Connect Custom Tabs


```
static void registerTabs() {
    CustomTabFactory::getInstance()->Register<IdentityMatchingTab
        >("identity_matching", "Shapes>>My␣Shape␣Tab␣1");

    CustomTabFactory::getInstance()->Register<
        ShapeInterpolationTab>("shape_interpolation", "
        Correspondences>>My␣Correspondences␣Tab␣1");
}
```



PETSc and SLEPc

- PETSc: (Sparse) linear algebra (e.g. matrix vector multiplication, linear systems, ...)
- SLEPc: (Sparse) eigenvalue solver (builds up on PETSc)
- Heavily used in the "Computational Science and Engineering" community for solving PDEs (e.g. Computational Fluid Dynamics)
- Based on MPI (therefore well suited for distributed HPCs)
- Can also be used with CUDA and cuBLAS (instead of MPI).
- Drawback: Coding is kinda' low level :(

PETSc and SLEPc

- PETSc: (Sparse) linear algebra (e.g. matrix vector multiplication, linear systems, ...)
- SLEPc: (Sparse) eigenvalue solver (builds up on PETSc)
- Heavily used in the "Computational Science and Engineering" community for solving PDEs (e.g. Computational Fluid Dynamics)
- Based on MPI (therefore well suited for distributed HPCs)
- Can also be used with CUDA and cuBLAS (instead of MPI).
- Drawback: Coding is kinda' low level :(

PETSc and SLEPc

- PETSc: (Sparse) linear algebra (e.g. matrix vector multiplication, linear systems, ...)
- SLEPc: (Sparse) eigenvalue solver (builds up on PETSc)
- Heavily used in the "Computational Science and Engineering" community for solving PDEs (e.g. Computational Fluid Dynamics)
- Based on MPI (therefore well suited for distributed HPCs)
- Can also be used with CUDA and cuBLAS (instead of MPI).
- Drawback: Coding is kinda' low level :(

PETSc and SLEPc

- PETSc: (Sparse) linear algebra (e.g. matrix vector multiplication, linear systems, ...)
- SLEPc: (Sparse) eigenvalue solver (builds up on PETSc)
- Heavily used in the "Computational Science and Engineering" community for solving PDEs (e.g. Computational Fluid Dynamics)
- Based on MPI (therefore well suited for distributed HPCs)
- Can also be used with CUDA and cuBLAS (instead of MPI).
- Drawback: Coding is kinda' low level :(

PETSc and SLEPc

- PETSc: (Sparse) linear algebra (e.g. matrix vector multiplication, linear systems, ...)
- SLEPc: (Sparse) eigenvalue solver (builds up on PETSc)
- Heavily used in the "Computational Science and Engineering" community for solving PDEs (e.g. Computational Fluid Dynamics)
- Based on MPI (therefore well suited for distributed HPCs)
- Can also be used with CUDA and cuBLAS (instead of MPI).
- Drawback: Coding is kinda' low level :(

PETSc and SLEPc

- PETSc: (Sparse) linear algebra (e.g. matrix vector multiplication, linear systems, ...)
- SLEPc: (Sparse) eigenvalue solver (builds up on PETSc)
- Heavily used in the "Computational Science and Engineering" community for solving PDEs (e.g. Computational Fluid Dynamics)
- Based on MPI (therefore well suited for distributed HPCs)
- Can also be used with CUDA and cuBLAS (instead of MPI).
- Drawback: Coding is kinda' low level :(

A Flavor of PETSc

$$\mathbf{b} := \mathbf{A} \cdot \mathbf{a}$$

```
// Initialize matrix A
int n = 10;
int m = 15;
Mat A;
MatCreateSeqDense(PETSC_COMM_SELF, n, m, NULL, &A);
for(PetscInt i = 0; i < n; i++) {
    for(PetscInt j = 0; j < m; j++) {
        PetscReal aij = rand();
        MatSetValue(A, i, j, aij, INSERT_VALUES);
    }
}
MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY);
```

A Flavor of PETSc

$$\mathbf{b} := \mathbf{A} \cdot \mathbf{a}$$

```
//Initialization of the vectors a and b
Vec a, b;
MatGetVecs(A, &a, &b);
for(PetscInt i = 0; i < size; i++) {
    VecSetValue(a, i, rand(), INSERT_VALUES);
}
VecAssemblyBegin(a);
VecAssemblyEnd(a);
```

A Flavor of PETSc

$$\mathbf{b} := \mathbf{A} \cdot \mathbf{a}$$

```
// Actual computation
MatMult(A, a, b);
VecView(b, PETSC_VIEWER_STDOUT_SELF);
MatDestroy(&A);
VecDestroy(&a);
VecDestroy(&b);
```

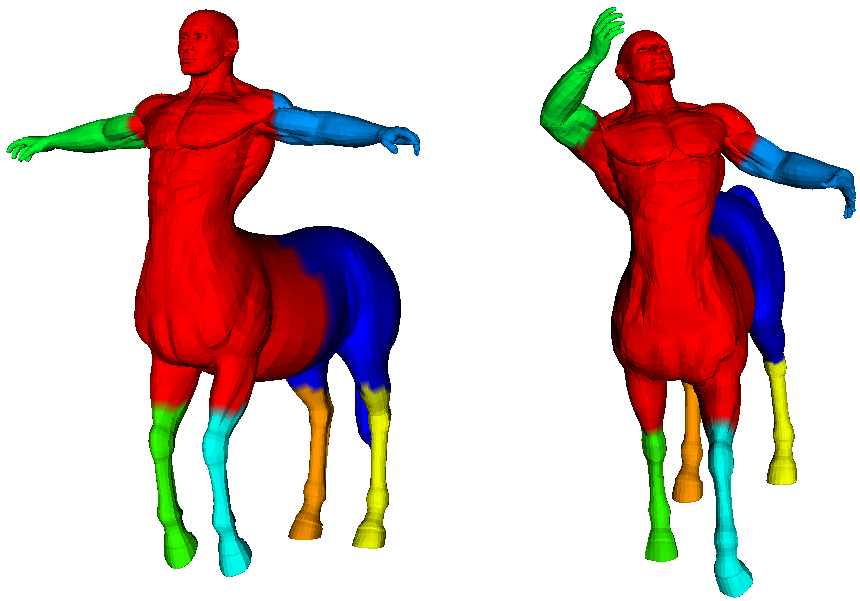
A Flavor of PETSc and CUDA

$$\mathbf{x} = a \cdot \mathbf{x} + \mathbf{y}$$

```
ierr = VecCUDACopyToGPU(xin);CHKERRQ(ierr);
ierr = VecCUDACopyToGPU(yin);CHKERRQ(ierr);
try {
    cusp::blas::axpy(*((Vec_CUDA*)xin->spptr)->GPUarray,
        *((Vec_CUDA*)yin->spptr)->GPUarray,
        alpha
    );
    yin->valid_GPU_array = PETSC_CUDA_GPU;
    ierr = WaitForGPU();
    CHKERRCUDA(ierr);
} catch(char *ex) { /*...*/ }
```

Functional maps: Intro

Goal: Transfer a function $f : V_M \to \mathbb{R}$ from shape M to shape N.
(Please note: $M = (V_M := \{v_1, v_2, \ldots, v_n\}, T_M)$ and therefore $f \in \mathbb{R}^n$)
Assumption: M and N are related by an *isometry*
(Ovsjanikov et al. 2012)



Functional maps: ONB

$$f = \sum_{i=1}^{n} a_i \phi_i \in \mathbb{R}^n$$

First guess (Canonical basis of $\mathbb{R}^n$): $\phi_i := \delta_{ij}$

Functional maps: ONB

$$f = \sum_{i=1}^{n} a_i \phi_i \in \mathbb{R}^n$$

First guess (Canonical basis of $\mathbb{R}^n$): $\phi_i := \delta_{ij}$

Question

Is this really a good idea? Answer: Nope!
Transferring a function f from M to N requires the correspondence Matrix $C \in \mathbb{R}^{n \times n}$ (But that is what we are looking for)

Functional maps: ONB

Better choice: Eigenvectors $\{\phi_i^M\}_{i=1}^n \subset \mathbb{R}^n$ and $\{\phi_i^N\}_{i=1}^n \subset \mathbb{R}^n$ of the Laplace-Beltrami operators $\Delta_M = M_M^{-1} \cdot L_M \in \mathbb{R}^{n \times n}$ and $\Delta_N = M_N^{-1} \cdot L_N \in \mathbb{R}^{n \times n}$

Functional maps: ONB

Property 1

If two shapes M, N are related by a perfect isometry, their Laplace-Beltrami eigenvectors are the same up two sign flip.

Functional maps: ONB

Property 1

If two shapes M, N are related by a perfect isometry, their Laplace-Beltrami eigenvectors are the same up two sign flip.

Property 2

If the signal f is sufficiently smooth we can represent it accurately (enough) by the first 20 eigenvectors:

$$f \approx \sum_{i=1}^{20} a_i \phi_i \in \mathbb{R}^n$$

Functional maps: ONB

Idea

Transfer f from M to N by projecting it onto Laplace-Beltrami eigenbasis $\{\phi_i^M\}_{i=1}^{20}$ and flip signs of coefficients accordingly:

$$\langle \phi_i^M, f \rangle_{M_M} = \langle \phi_i^M, \sum_{j=1}^{20} a_j \phi_j^M \rangle_{M_M} = \sum_{j=1}^{20} a_j \langle \phi_i^M, \phi_j^M \rangle_{M_M} = a_i$$

$$t(f) = \sum_{i=1}^{20} b_i \phi_i^N = \sum_{i=1}^{20} \underbrace{c_{ii} a_i}_{b_i} \phi_i^N \quad \text{with } c_{ii} \in \{-1, 1\}$$

Functional maps: ONB

Idea

Transfer f from M to N by projecting it onto Laplace-Beltrami eigenbasis $\{\phi_i^M\}_{i=1}^{20}$ and flip signs of coefficients accordingly:

$$\langle \phi_i^M, f \rangle_{M_M} = \langle \phi_i^M, \sum_{j=1}^{20} a_j \phi_j^M \rangle_{M_M} = \sum_{j=1}^{20} a_j \langle \phi_i^M, \phi_j^M \rangle_{M_M} = a_i$$

$$t(f) = \sum_{i=1}^{20} b_i \phi_i^N = \sum_{i=1}^{20} \underbrace{c_{ii} a_i}_{b_i} \phi_i^N \quad \text{with } c_{ii} \in \{-1, 1\}$$

$$t(f) = \underbrace{C \cdot a}_{=b} \cdot \Phi^N \quad \text{with } C \in \mathbb{R}^{20 \times 20}, C \text{ diagonal}$$

Functional maps: Idea

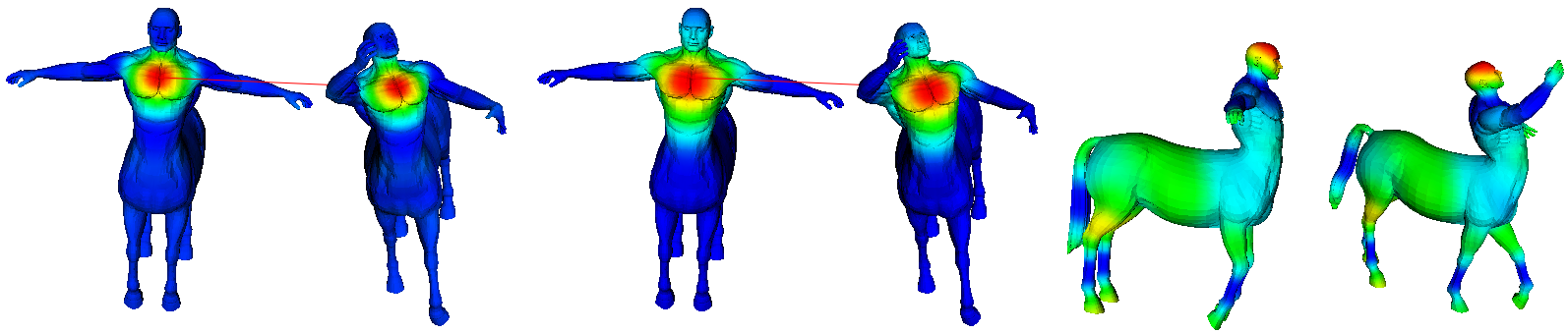
Caveat: Since we never deal with perfect isometries, C is not perfectly diagonal:

C

Functional maps: Compute C

Technische Universität München

Assume we are given a set of corresponding constraint functions represented as 20-dimensional vectors of coefficients $a_i^\top, b_i^\top \in \mathbb{R}^{20}$:

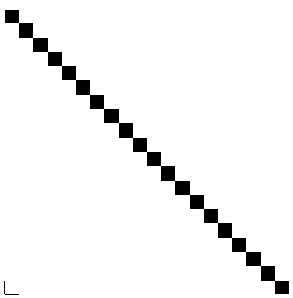


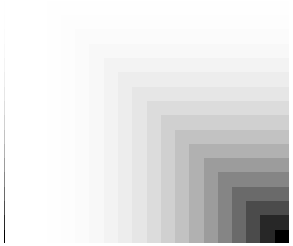
$$\min_C \underbrace{\frac{1}{2}\|B - CA\|_F^2}_{\text{data term}} + \lambda \underbrace{|W \circ C|_1}_{\text{regularizer}}$$

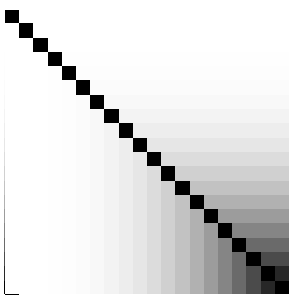
Functional maps: Define W

First guess: Squared distance from diagonal: $W_{ij} := 10 \cdot (i-j)^2$
Better:

$$W_{ij} := \min\{|i-j|, 1\} \cdot (100 - \left(\frac{\min\{i,j\}}{20}\right)^3 \cdot 100)$$







Functional maps: Outliers

$$\min_{C,O} \frac{1}{2}\|B - CA - O\|_F^2 + \lambda|W \circ C|_1 + \mu\|O\|_{2,1}$$

where $\|O\|_{2,1} := \sum_{i=1}^{I} \|o_i^\top\|_2$ (Pokrass et al. 2013)

Functional maps: Optimization

$$\min_{C,O} \underbrace{\frac{1}{2}\|B - CA - O\|_F^2}_{\text{differentiable}} + \underbrace{\lambda|W \circ C|_1 + \mu\|O\|_{2,1}}_{\text{not differentiable}}$$

- Whole function convex
- Optimization by Forward-Backward Splitting (Generalized projected gradient descent)

A Function Transfer Tab

Demo time

Questions?

Thank you for your attention!