camp Documentation

Blake Zimmerman

Feb 16, 2021

CONTENTS

CAMP is designed as a general-purpose tool for medical image and data processing. PyTorch provides an efficient backend for mathematical routines (linear algebra, automatic differentiation, etc.) with support for GPU acceleration and distributed computing. PyTorch also allows CAMP to be portable due to full CPU-only support.

CAMP adopts coordinate system conventions designed to be compatible with medical imaging systems, including meta-data for a consistent world coordinate frame. Image transformations and image processing can be performed relative to the coordinate system in which the images reside, which facilitates multi-scale algorithms. Core representations for data include structured and unstructured grids.

CHAPTER

ONE

STRUCTURED GRIDS

Structured grids represent data with consistent rectilinear element spacing. These grids are commonly defined by the origin and element spacing attributes. Images and vector fields are most commonly represented by a structured grid. CAMP defines many operators to perform computation on structured grid data, including *Gaussian blur* and *composition of deformation fields*. Multi-channel (including color) images are supported – the internal convention is channels-first representation (C x D x H x W).

CHAPTER

TWO

UNSTRUCTURED GRIDS

Unstructured grids represent data with arbitrary element shape and location. Currently, only triangular mesh objects are supported, which aim to represent surface data via edge and vertex representation. Data values may be face-centered or node-centered. The unstructured grid objects maintain a world coordinate system that preserves relationships between other unstructured and structured grid data. An example implementation of deformable surface-based registration is implemented using the unstructured grid representation, based on Glaunes et al. (2004). Watch a summary video using this implementation on YouTube.

CHAPTER

THREE

DATA I/O

Many medical imaging formats are supported through SimpleITK.

CHAPTER

FOUR

RELEVANT PUBLICATIONS

There is not a single publication that describes the architecture or design of the CAMP project. However, the following publications are use cases that inspired the core development of CAMP.

- **Zimmerman, B. E.**, Johnson, S. L., Odéen, H. A., Shea, J. E., Factor, R. E., Joshi, S. C., & Payne, A. H. (2021). Histology to 3D In Vivo MR Registration for Volumetric Evaluation of MRgFUS Treatment Assessment Biomarkers. *Manuscript submitted for publication*.
- **Zimmerman, B. E.**, Johnson, S., Odéen, H., Shea, J., Foote, M. D., Winkler, N., Sarang Joshi, & Payne, A. (2020). Learning Multiparametric Biomarkers for Assessing MR-Guided Focused Ultrasound Treatments. IEEE Transactions on Biomedical Engineering.

CHAPTER

FIVE

TABLE OF CONTENTS

5.1 Core

This section details the currently implemented base classes available in CAMP, including the Structured Grid and Triangle Mesh objects. The structured grids represent images, fields, look-up tables, or anything that is structured on a grid. The triangle mesh object inherits from an underlying unstructured grid object meant to represent different surfaces. Currently, the only mesh type supported is a triangle mesh, but the unstructured grid object could easily be expanded to include other mesh types, such as quads. This package also provides functions for displaying both structured grid data and triangle mesh objects.

5.1.1 Structured Grid

class StructuredGrid(*size*, *spacing=None*, *origin=None*, *device='cpu'*, *dtype=torch.float32*, *requires_grad=False*, *tensor=None*, *channels=1*)

This is the base class for grid structured data such as images, look-up tables (luts), vector fields, etc. This class wraps a torch tensor (data attribute) to provide world coordinate system context.

Parameters

- **size** (*list, tuple, tensor*) – Size of the grid. Size is ordered [z],y,x ([] is optional).
- **spacing** (*list, tuple, tensor, optional*) – Spacing between the grid elements. Default is isotropic 1.0 spacing.
- **origin** (*list, tuple, tensor, optional*) – Real world location of the pixel (2D) or voxel (3D) with the minimum location value. The locations of the grid elements increase by the spacing in each relative direction from this voxel. Default pleaces the center of the grid at the origin.
- **device** (*str, optional*) – Memory location - one of 'cpu', 'cuda', or 'cuda:X' where X specifies the device identifier. Default: 'cpu'
- **dtype** (*str, optional*) – Data type, specified from torch memory types. Default: 'torch.float32'
- **requires_grad** (*bool, optional*) – Track tensor for gradient operations. Default: False
- **tensor** (*torch.tensor, optional*) – The underlying tensor for the data attribute. This allows `StructuredGird` to be wrapped around alread-exisiting tensors. This tensor must be of size C,[z],y,x where [z],y,x are the same as 'size' and C is equal to Channels. If not provided, the data attribute will be initialized to size C,[z],y,x with zeros.

- **channels** (*int*) – Number of channels for the grid. For example, black and white images must have 1 channel and RGB images have 3 channels. Channels can be any integer number.

static FromGrid(*grid*, *tensor=None*, *channels=1*)
Construct a new *StructuredGrid* from a reference *StructuredGrid* (for the size, spacing, origin, device, dtype, requires_grad) and a torch tensor.

Parameters

- **grid** (*StructuredGrid*) – Reference *StructuredGrid* with the reference attributes.
- **tensor** (*tensor*) – Torch tensor to wrap into the new *StructuredGrid*. Must have size [z],y,x from the reference *StructuredGrid* and the number of specific channels.
- **channels** (*int*) – Channels of the input tensor.

Returns New *StructuredGrid* wrapped around the input tensor.

clone()
Create a copy of the *StructuredGrid*.

Returns Copy of *StructuredGrid*.

copy()
Create a copy of the *StructuredGrid*.

Returns Copy of *StructuredGrid*.

extract_slice(*index*, *dim*)
Extract a slice from a 3D volume. Updates the origin to maintain the world coordinate system location.

Parameters

- **index** (*int*) – Slice index to extract.
- **dim** (*int*) – Dimension along which to extract the slice.

Returns Extracted slice.

get_subvol(*zrng=None*, *yrng=None*, *xrng=None*)
Extract a sub volume. The coordiantes for the sub volume are in index coordiantes. Updates the origin to maintain the world coordinate system location.

Parameters

- **zrng** (*list, tuple, optional*) – Tuple or list of 2 values between [0, size[0]]. If no range is provided, the size stays the same.
- **yrng** (*list, tuple, optional*) – Tuple or list of 2 values between [0, size[1]]. If no range is provided, the size stays the same.
- **xrng** (*list, tuple, optional*) – Tuple or list of 2 values between [0, size[2]]. If no range is provided, the size stays the same.

Returns Sub volume with updated origin.

max()
Max of the data attribute.

Returns data.max()

min()
Min of the data attribute.

Returns data.min()

minmax()
Min and Max of the data attribute.

Returns [data.min(), data.max()]

set_origin_(*origin*)
Set the origin. Does not change the spacing.

Parameters **origin** (*list, tuple, tensor*) – New origin.

Returns None

set_size(*size*, *inplace=True*)
Set the size of the *StructuredGrid*. This will update the spacing and origin of the *StructuredGrid* to maintain the original real world FOV.

Parameters

- **size** (*torch.tensor*) – New size for the *StructuredGrid* [z],y,x
- **inplace** (*bool*) – Perform the resize operation in place. Default=True.

Returns If inplace==True then returns a new *StructuredGrid*.

set_spacing_(*spacing*)
Set the spacing. Does not change the origin.

Parameters **spacing** (*list, tuple, tensor,*) – New spacing.

Returns None

set_to_identity_lut_()
Set the tensor to an real world identity look-up table (LUT) using the spacing and origin of the *StructuredGrid*. The number of channels will be set to the number of dimensions in size.

Returns *StructuredGrid* as a real world identity LUT.

shape()
Returns the shape of the data attribute, including the channels.

Returns data.shape

sum()
Sum of the data attribute.

Returns data.sum()

to_(*device*)
Change the memory device of the *StructuredGrid*.

Parameters **device** (*str, optional*) – New memory location - one of 'cpu', 'cuda', or 'cuda:X' where X specifies the device identifier.

Returns None

to_type_(*new_type*)
Change the data type of the *StructuredGrid* attributes.

Parameters **dtype** (*str, optional*) – Data type, specified from torch memory types. Default: 'torch.float32'

Returns None

5.1.2 Triangle Mesh

class TriangleMesh(*vertices*, *indices*, *per_vert_values=None*, *per_index_values=None*)
Triangle mesh class that inherits from the unstructured grid class.

Parameters

- **vertices** (*tensor*) – Vertices of the mesh object (x,y,z)
- **indices** (*long tensor*) – Indices of the mesh object
- **per_vert_values** (*tensor, optional*) – Values associated with each vertex of the triangle mesh.
- **per_index_values** (*str, optional*) – Values associated with the indices (or faces) of the triangle mesh.

add_surface_(*verts*, *indices*)
Concatenate two triangle mesh objects. This does not connect the two objects with faces, it just concatenates the vertices and indices of the two surfaces into one.

Returns None

calc_centers(***kwargs*)
Caluclate the face centers of the triangle mesh using the vertices and indices to populate the centers attribute.

Returns None

calc_normals()
Caluclate the face normals of the triangle mesh using the vertices and indices to populate the normals attribute.

Returns None

flip_normals_()
Flip the face normals.

Returns None

5.1.3 Display

DispFieldGrid(*Field*, *grid_size=None*, *title=None*, *newFig=True*, *dim='z'*, *slice_index=None*)
Displays a grid of the input field. Field is assumed to be a look-up table (LUT) of type `StructuredGrid`.

Parameters

- **Field** (`StructuredGrid`) – Assumed to be a `StructuredGrid` LUT that defines a transformation.
- **grid_size** (*int*) – Number of grid lines to plot in each direction.
- **title** (*str*) – Figure Title.
- **newFig** (*bool*) – Create a new figure. Default True.
- **dim** (*str*) – Dimension along which to plot 3D image. Default is 0 ('z').
- **slice_index** (*int*) – Slice index along 'dim' to plot

Returns None

DispImage(*Image*, *rng=None*, *cmap='gray'*, *title=None*, *new_figure=True*, *color=False*, *colorbar=True*, *axis='default'*, *dim=0*, *slice_index=None*)

Display an image default with a colorbar. If the input image is 3D, it will be sliced along the dim argument. If no slice index is provided then it will be the center slice along dim.

Parameters

- **Image** (StructuredGrid) – Input Image ([RGB[A]], [Z], Y, X)
- **rng** (*list, tuple*) – Display intensity range. Defaults to data intensity range.
- **cmap** (*str*) – Matplotlib colormap. Default 'gray'.
- **title** (*str*) – Figure Title.
- **new_figure** (*bool*) – Create a new figure. Default True.
- **colorbar** (*bool*) – Display colorbar. Default True.
- **axis** (*str*) – Axis direction. 'default' has (0,0) in the upper left hand corner and the x direction is vertical 'cart' has (0,0) in the lower left hand corner and the x direction is horizontal
- **dim** (*int*) – Dimension along which to plot 3D image. Default is 0 (z).
- **slice_index** (*int*) – Slice index along 'dim' to plot

Returns None

exception DisplayException

exception for this class

DisplayJacobianDeterminant(*Field*, *rng=None*, *cmap='jet'*, *title=None*, *new_figure=True*, *colorbar=True*, *slice_index=None*, *dim='z'*)

Calculated and display the jacobian determinant of a field.

Parameters

- **Field** (StructuredGrid) – Assumed to be a StructuredGrid LUT that defines a transformation.
- **rng** (*list, tuple*) – Display intensity range. Defaults to jacobian determinant intensity range.
- **cmap** (*str*) – Matplotlib colormap. Default 'jet'.
- **title** (*str*) – Figure Title.
- **new_figure** (*bool*) – Create a new figure. Default True.
- **colorbar** (*bool*) – Display colorbar. Default True.
- **dim** (*int*) – Dimension along which to plot 3D image. Default is 0 (z).
- **slice_index** (*int*) – Slice index along 'dim' to plot

Returns None

EnergyPlot(*energy*, *title='Energy'*, *new_figure=True*, *legend=None*)

Plot energies from registration functions.

Parameters

- **energy** (*list, tuple*) – The energies should be in the form [E1list, E2list, E3list, ...]
- **title** (*str*) – Figure Title.
- **new_figure** (*bool*) – Create a new figure. Default True.

- **legend** (*list*) – List of strings to be added to the legend in the form [E1legend, E2legend, E3legend, . . .]

Returns None

PlotSurface (*verts*, *faces*, *fig=None*, *norms=None*, *cents=None*, *ax=None*, *color=(0, 0, 1)*)
Plot a triangle mesh object.

Parameters

- **verts** (*tensor*) – Vertices of the mesh object.
- **faces** (*tensor*) – Indices of the mesh object.
- **fig** (*Maplotlib figure object*) – Matplotlib figure object to plot the surface on. If one is not provided, and new one is created.
- **norms** (*tensor, optional*) – Normals of the mesh object.
- **cents** (*tensor, optional*) – Centers of the mesh object.
- **ax** (*Maplotlib axis object*) – Matplotlib axis object to plot the surface on. If one is not provided, and new one is created.
- **color** (*tuple*) – Plotted color of the surface. Tuple of three floats between 0 and 1 specifying RGB values.

Returns None

5.2 File Input/Output (IO)

5.2.1 ITK IO

LoadITKFile (*filename*, *device='cpu'*, *dtype=torch.float32*)
Load an ITK compatible file using the SimpleITK package into a `StructuredGrid` object.

Parameters

- **filename** (*str*) – File path
- **device** (*str, optional*) – Memory location - one of 'cpu', 'cuda', or 'cuda:X' where X specifies the device identifier. Default: 'cpu'
- **dtype** (*str, optional*) – Data type, specified from torch memory types. Default: 'torch.float32'

Returns `StructuredGrid`

SaveITKFile (*grid*, *f_name*)
Save a `StructuredGrid` object to an ITK compatible file using the SimpleITK package.

Parameters

- **grid** – `StructuredGrid` to be saved.
- **f_name** (*str*) – File path

Returns None

5.2.2 Object IO

ReadOBJ(*file*, *device='cpu'*)
Read a triangle mesh OBJ file into vertex and index tensors.

Parameters

- **file** (*str*) – File path
- **device** (*str, optional*) – Memory location - one of 'cpu', 'cuda', or 'cuda:X' where X specifies the device identifier. Default: 'cpu'

Returns [Vertices, Faces]

WriteOBJ(*vert*, *faces*, *file*)
Write a triangle mesh object defined by verts and faces to an OBJ file.

Parameters

- **vert** (*tensor*) – Output vertices (must be on the CPU).
- **faces** (*tensor*) – Output indices (must be on the CPU).
- **file** (*str*) – OBJ file to be written.

Returns None

5.3 Structured Grid Operators

5.3.1 Binary Operators

Affine Intensity Filter

class AffineIntensity(*similarity*, *dim=2*, *init_affine=None*, *init_translation=None*, *device='cpu'*, *dtype=torch.float32*)

static Create(*similarity*, *dim=2*, *init_affine=None*, *init_translation=None*, *device='cpu'*, *dtype=torch.float32*)
Object for registering two structured grids with an affine transformation. The affine and translation are optimized independently. This Affine Intensity filter must be on the same device as the target and moving structured grids. The affine and translation attributes have requires_grad=True so they can be added to a torch optimizer and updated with autograd functions.

Parameters

- **similarity** (Filter) – This is the similiarty filter used to compare the two structured grids to be registered. This filter is usually a Binary Operator (ie. L2 Image Similarity)
- **dim** (*int*) – Dimensionality of the structured grids to be registered (not including channels).
- **init_affine** (*tensor, optional*) – Initial affine to apply to the moving structured grid.
- **init_translation** (*tensor, optional*) – Initial translation to apply to the moving structured grid.
- **device** (*str*) – Memory location - one of 'cpu', 'cuda', or 'cuda:X' where X specifies the device identifier. Default: 'cpu'

- **dtype** (*str*) – Data type for the attributes. Specified from torch memory types. Default: 'torch.float32'

Returns Affine Intensity Filter Object

forward(*target*, *moving*)

Apply the forward affine operation applied to the moving image and calculate the resulting similarity measure between the target and moving images. The gradients on the affine and translation attributes are tracked through this forward operation so that the gradient update can be applied to update the affine and translation. This function is meant to be used iteratively in the registration process.

Parameters

- **target** (StructuredGrid) – Target structured grid. Does not get updated or changed.
- **moving** (StructuredGrid) – Moving structured grid. Affine and translation are applied this structured grid before the similarity calculation.

Returns Energy from the similarity evaluation (usually a single float).

Compose Grids Filter

class ComposeGrids(*padding_mode='border'*, *device='cpu'*, *dtype=torch.float32*)

static Create(*padding_mode='border'*, *device='cpu'*, *dtype=torch.float32*)

Object to compose StructuredGrid look-up table fields into one grid.

Parameters

- **pad_mode** (*str*) – padding mode for outside grid values - one of 'zeros', 'border', or 'reflection'. Default: 'zeros'
- **device** (*str*) – Memory location - one of 'cpu', 'cuda', or 'cuda:X' where X specifies the device identifier. Default: 'cpu'
- **dtype** (*str*) – Data type for the attributes. Specified from torch memory types. Default: 'torch.float32'

Returns Object to compose a list of look-up tables.

forward(*L*)

Given a list of StructuredGrid look-up tables L = [L0, L1, L2] returns a composed look-up table comp_field = L2(L1(L0(x))) of type StructuredGrid.

Parameters **L** (*list, tuple*) – List of look-up tables. All fields in the list must be on the same memory device.

Returns Composed look-up tables Comp_filed

L2 Image Similarity

class L2Similarity(*dim=2*, *device='cpu'*, *dtype=torch.float32*)

static Create(*dim=2*, *device='cpu'*, *dtype=torch.float32*)
Compare two `StructuredGrid` objects using an L2 similarity metric.

Parameters

- **dim** (*int*) – Dimensionality of the `StructuredGrid` to be compared (not including channels).
- **device** (*str*) – Memory location - one of 'cpu', 'cuda', or 'cuda:X' where X specifies the device identifier. Default: 'cpu'
- **dtype** (*str*) – Data type for the attributes. Specified from torch memory types. Default: 'torch.float32'

Returns L2 comparision object.

c1(*target*, *moving*, *grads*)
First derivative of the L2 similarity metric.

Parameters

- **target** (`StructuredGrid`) – Structured Grid 1
- **moving** (`StructuredGrid`) – Structured Grid 2
- **grads** (`StructuredGrid`) – Gradients of the moving image.

Returns

forward(*target*, *moving*)
Compare two `StructuredGrid` with L2 similarity metric. This is often used for registration so the variables are labeled as target and moving. This function preserves the dimensionality of the original grids.

Parameters

- **target** (`StructuredGrid`) – Structured Grid 1
- **moving** (`StructuredGrid`) – Structured Grid 2

Returns L2 similarity as `StructuredGrid`

Normalized Cross Correlation Filter

class NormalizedCrossCorrelation(*grid*, *window=5*, *device='cpu'*, *dtype=torch.float32*)

static Create(*grid*, *window=5*, *device='cpu'*, *dtype=torch.float32*)

c1(*target*, *moving*, *grads*)

forward(*target*, *moving*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

5.3.2 Unary Operators

Affine Transform Filter

class AffineTransform(*target_landmarks=None*, *source_landmarks=None*, *affine=None*, *rigid=False*, *interp_mode='bilinear'*, *device='cpu'*, *dtype=torch.float32*)

Bases: `camp.StructuredGridOperators.UnaryOperators._UnaryFilter.Filter`

static Create(*target_landmarks=None*, *source_landmarks=None*, *affine=None*, *rigid=False*, *interp_mode='bilinear'*, *device='cpu'*, *dtype=torch.float32*)

Returns an Affine Transform Filter that can be applied to type `StructuredGrid`. This can be initiated using a pair of landmarks (target and source) or with a pre-defined affine transformation (affine). Either both target and source landmarks must be provided OR a pre-defined affine.

Parameters

- **target_landmarks** (*tensor, optional*) – Target or unmoving landmarks selected in the target space. This tensor should be of size Nxdim where N is the number of landmarks and dim is the dimensionality of the `StructuredGrid` the affine will be applied to.
- **source_landmarks** (*tensor, optional*) – Source or moving landmarks selected in the source space. This tensor should be of size Nxdim where N is the number of landmarks and dim is the dimensionality of the `StructuredGrid` the affine will be applied to.
- **affine** (*tensor, optional*) – Pre-defined affine. This should be of shape (dim + 1)x(dim + 1) where the added dimension stores the translation.
- **rigid** (*bool*) – If the affine should be reduced to rigid transform only. Default is False.
- **interp_mode** (*str*) – Resampling interpolation mode to be used when applying the defromation - one of 'bilinear' or 'nearest'. Default: 'bilinear'
- **device** (*str*) – Memory location for the created filter - one of 'cpu', 'cuda', or 'cuda:X' where X specifies the device identifier. Default: 'cpu'
- **dtype** (*str*) – Data type for the filter attributes. Specified from torch memory types. Default: 'torch.float32'

Note: When mode='bilinear' and the input is 5-D, the interpolation mode used internally will actually be trilinear. However, when the input is 4-D, the interpolation mode will legitimately be bilinear.

Returns Affine transform filter object with the specified parameters.

forward(*x*, *out_grid=None*, *xyz_affine=False*)

Resamples the `Core.StructuredGrid` through the affine attribute onto the same grid or the out_grid if out_grid is provided.

Parameters

- **x** (Core.StructuredGrid) – StructuredGrid to be transformed by the affine attribute.
- **out_grid** (Core.StructuredGrid, optional) – An optional additional grid that specifies the output grid. If not specified, the output grid will be the same as the input grid (x).
- **xyz_affine** (*bool, optional*) – Is affine xyz ordered instead of zyx?

Returns Affine transformed StructredGrid

Apply Grid Filter

class ApplyGrid(*grid*, *interp_mode='bilinear'*, *pad_mode='zeros'*, *device='cpu'*, *dtype=torch.float32*)

Bases: camp.StructuredGridOperators.UnaryOperators._UnaryFilter.Filter

static Create(*grid*, *interp_mode='bilinear'*, *pad_mode='zeros'*, *device='cpu'*, *dtype=torch.float32*)

Returns an Apply Grid Filter that contained a deformation field that can be applied to type StructuredGrid and adds all attributes to the appropriate memory device.

Parameters

- **grid** (StructuredGrid) – The deformation field to be applied by the Apply Grid Filter. This is assumed to be in real-world coordinates relative to the spacing and origin of the grid.
- **interp_mode** (*str*) – Resampling interpolation mode to be used when applying the defromation - one of 'bilinear' or 'nearest'. Default: 'bilinear'
- **pad_mode** (*str*) – padding mode for outside grid values - one of 'zeros', 'border', or 'reflection'. Default: 'zeros'
- **device** (*str*) – Memory location for the created Apply Grid Filter - one of 'cpu', 'cuda', or 'cuda:X' where X specifies the device identifier. Default: 'cpu'
- **dtype** (*str*) – Data type for the Apply Grid Filter attributes. Specified from torch memory types. Default: 'torch.float32'

Note: When mode='bilinear' and the input is 5-D, the interpolation mode used internally will actually be trilinear. However, when the input is 4-D, the interpolation mode will legitimately be bilinear.

Returns Apply Grid Filter with the specified parameters.

forward(*in_grid*, *out_grid=None*)

Apply the grid attribute to in_grid.

Parameters

- **in_grid** (StructuredGrid) – The :class:'StructuredGrid' to apply the grid attribute to.
- **out_grid** (StructuredGrid, optional) – An optional additional grid that specifies the output grid. If not specified, the output grid will be the same as the input grid.

Returns Returns in_grid resampled through the grid attribute onto the out_grid.

Divergence Filter

class Divergence(*dim=2*, *device='cpu'*, *dtype=torch.float32*)

static Create(*dim=2*, *device='cpu'*, *dtype=torch.float32*)
Create a object to calculate the divergence of a look-up table field `StructuredGrid`.

Parameters

- **dim** (*int*) – Dimension of the `StructuredGrid` the filter will be applied to (not including channels).
- **device** (*str*) – Memory location for the created filter - one of 'cpu', 'cuda', or 'cuda:X' where X specifies the device identifier. Default: 'cpu'
- **dtype** (*str*) – Data type for the filter attributes. Specified from torch memory types. Default: 'torch.float32'

Returns Divergence filter object with the specified parameters.

forward(*x*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

Fluid Kernel Filter

class FluidKernel(*grid*, *alpha=1.0*, *beta=0.0*, *gamma=0.001*, *device='cpu'*, *dtype=torch.float32*)

static Create(*grid*, *alpha=1.0*, *beta=0.0*, *gamma=0.001*, *device='cpu'*, *dtype=torch.float32*)

apply_forward(*x*)

apply_inverse(*x*)

forward(*x*, *inverse*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

project_incompressible(*x*)

set_size(*grid*)

Gaussian Filter

class Gaussian(*channels*, *kernel_size*, *sigma*, *dim=2*, *device='cpu'*, *dtype=torch.float32*)
Bases: camp.StructuredGridOperators.UnaryOperators._UnaryFilter.Filter

static Create(*channels*, *kernel_size*, *sigma*, *dim=2*, *device='cpu'*, *dtype=torch.float32*)
Create a filter to gaussian blur a StructuredGrid with the specified number of channels.

Parameters

- **channels** (*int*) – Number of channels in the StructuredGrid to be blurred.
- **kernel_size** (*int*) – Size of the kernel to use.
- **sigma** (*int*) – Sigma of the gaussian kernel.
- **dim** (*int*) – Number of dimensions in the StructuredGrid the filter will be applied to (not including channels).
- **device** (*str*) – Memory location for the created filter - one of 'cpu', 'cuda', or 'cuda:X' where X specifies the device identifier. Default: 'cpu'
- **dtype** (*str*) – Data type for the filter attributes. Specified from torch memory types. Default: 'torch.float32'

Returns Gaussian filter object with the specified parameters.

forward(*x*)
Apply the gaussian filter to the input StructuredGrid x.

Parameters **x** (StructuredGrid) – StructuredGrid to apply the gaussian to.

Returns Gaussian filtered StructuredGrid.

Gradient Filter

class Gradient(*dim=2*, *device='cpu'*, *dtype=torch.float32*)
Bases: camp.StructuredGridOperators.UnaryOperators._UnaryFilter.Filter

static Create(*dim=2*, *device='cpu'*, *dtype=torch.float32*)
Create a filter to calculate the central difference of a StructuredGrid.

Parameters

- **dim** (*int*) – Number of dimensions in the StructuredGrid the filter will be applied to (not including channels).
- **device** (*str*) – Memory location for the created filter - one of 'cpu', 'cuda', or 'cuda:X' where X specifies the device identifier. Default: 'cpu'
- **dtype** (*str*) – Data type for the filter attributes. Specified from torch memory types. Default: 'torch.float32'

Returns Gaussian filter object with the specified parameters.

forward(*x*)
Calculate the gradient of the input StructuredGrid x.

Parameters **x** (StructuredGrid) – StructuredGrid to calculate the gradients of.

Returns Gradients of the input StructuredGrid.

Gradient Regularizer

class NormGradient(*weight*, *dim=2*, *device='cpu'*, *dtype=torch.float32*)

static Create(*weight*, *dim=2*, *device='cpu'*, *dtype=torch.float32*)

forward(*vector_field*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

Jacobian Determinant Filter

class JacobianDeterminant(*dim=2*, *device='cpu'*, *dtype=torch.float32*)

static Create(*dim=2*, *device='cpu'*, *dtype=torch.float32*)
Create a filter to calculate the Jacobian determinant of a look-up table `StructuredGrid`.

Parameters

- **dim** (*int*) – Number of dimensions in the `StructuredGrid` the filter will be applied to (not including channels).
- **device** (*str*) – Memory location for the created filter - one of 'cpu', 'cuda', or 'cuda:X' where X specifies the device identifier. Default: 'cpu'
- **dtype** (*str*) – Data type for the filter attributes. Specified from torch memory types. Default: 'torch.float32'

Returns Jacobian determinant filter object.

forward(*x*)
Calculate the Jacobian determinant of the input `StructuredGrid` x.

Parameters **x** (`StructuredGrid`) – `StructuredGrid` to calculate the Jacobian determinant of.

Returns Jacobian determinant of the input `StructuredGrid`.

Radial Basis Filter

Resample World Filter

class ResampleWorld(*grid*, *interp_mode='bilinear'*, *pad_mode='zeros'*, *device='cpu'*, *dtype=torch.float32*)

static Create(*grid*, *interp_mode='bilinear'*, *pad_mode='zeros'*, *device='cpu'*, *dtype=torch.float32*)

forward(*x*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

Variance Equalize Filter

class VarianceEqualize(*kernel_size=11*, *sigma=2.0*, *eps=0.001*, *device='cpu'*, *dtype=torch.float32*)

Takes an Image and gives the variance equalized version.

I_out, Im: PyCA Image3Ds sigma: (scalar) gaussian filter parameter eps: (scalar) division regularizer

sigma is the width (in voxels) of the gaussian kernel eps is the regularizer

for a gaussian kernel k, we have

I_ve = I'/sqrt(k * I'^2) where I' = I - k*I

static Create(*kernel_size=11*, *sigma=2.0*, *eps=0.001*, *device='cpu'*, *dtype=torch.float32*)

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

5.4 Structured Grid Tools

5.4.1 Gradient Flow Filter

class IterativeMatch(*source*, *target*, *similarity*, *operator*, *regularization=None*, *step_size=0.001*, *incompressible=True*, *device='cpu'*, *dtype=torch.float32*)

Bases: `camp.StructuredGridTools._BaseTool.Filter`

static Create(*source*, *target*, *similarity*, *operator*, *regularization=None*, *step_size=0.001*, *incompressible=True*, *device='cpu'*, *dtype=torch.float32*)

energy()

get_field()

get_image()

step()

5.5 Unstructured Grid Operators

5.5.1 Binary Operators

Affine Currents

class AffineCurrents(*tar_normals*, *tar_centers*, *sigma*, *init_affine=None*, *init_translation=None*, *kernel='cauchy'*, *device='cpu'*, *dtype=torch.float32*)

Bases: `torch.nn.modules.module.Module`

static Create(*tar_normals*, *tar_centers*, *sigma*, *init_affine=None*, *init_translation=None*, *kernel='cauchy'*, *device='cpu'*, *dtype=torch.float32*)

forward(*src_normals*, *src_centers*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

Currents Energy Filter

class CurrentsEnergy(*tar_normals*, *tar_centers*, *sigma*, *kernel='cauchy'*, *device='cpu'*, *dtype=torch.float32*)

Bases: `torch.nn.modules.module.Module`

static Create(*tar_normals*, *tar_centers*, *sigma*, *kernel='cauchy'*, *device='cpu'*, *dtype=torch.float32*)

static cauchy(*d*, *sigma*)

static colordiff(*src_colors*, *tar_colors*)

static distance(*src_centers*, *tar_centers*)

forward(*src_normals*, *src_centers*, *tar_normals*, *tar_centers*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

static gaussian(*d*, *sigma*)

Deformable Currents Filter

class DeformableCurrents(*src_surface*, *tar_surface*, *sigma*, *kernel='cauchy'*, *device='cpu'*, *dtype=torch.float32*)

Bases: `torch.nn.modules.module.Module`

static Create(*src_surface*, *tar_surface*, *sigma*, *kernel='cauchy'*, *device='cpu'*, *dtype=torch.float32*)

forward()

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

Single Angle Affine Currents Filter

class SingleAngleCurrents(*tar_normals*, *tar_centers*, *sigma*, *init_angle=None*, *init_translation=None*, *kernel='cauchy'*, *device='cpu'*, *dtype=torch.float32*)

Bases: `torch.nn.modules.module.Module`

static Create(*tar_normals*, *tar_centers*, *sigma*, *init_angle=None*, *init_translation=None*, *kernel='cauchy'*, *device='cpu'*, *dtype=torch.float32*)

build_matrix()

forward(*src_normals*, *src_centers*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

Stitching Currents Filter

class StitchingCurrents(*src_surface*, *tar_surface*, *reference_surface*, *sigma*, *kernel='cauchy'*, *device='cpu'*, *dtype=torch.float32*)

Bases: `torch.nn.modules.module.Module`

static Create(*src_surface*, *tar_surface*, *ref_surface*, *sigma*, *kernel='cauchy'*, *device='cpu'*, *dtype=torch.float32*)

static cauchy(*d*, *sigma*)

static distance(*src_centers*, *tar_centers*)

energy(*src_normals*, *src_centers*, *tar_normals*, *tar_centers*)

forward()

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

static gaussian(*d*, *sigma*)

5.5.2 Unary Operators

Affine Transform Filter

class AffineTransformSurface(*affine*, *rigid=False*, *device='cpu'*, *dtype=torch.float32*)

Bases: `camp.UnstructuredGridOperators.UnaryOperators._UnaryFilter.Filter`

static Create(*affine*, *rigid=False*, *device='cpu'*, *dtype=torch.float32*)

forward(*obj_in*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

Gaussian Smoothing Filter

class GaussianSmoothing(*sigma*, *dim=2*, *device='cpu'*, *dtype=torch.float32*)

Bases: `camp.UnstructuredGridOperators.UnaryOperators._UnaryFilter.Filter`

static Create(*sigma*, *dim=2*, *device='cpu'*, *dtype=torch.float32*)

forward(*verts*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

D

E

F

G

I

J

L

V

W