Geometry Processing (601.458/658)

Misha Kazhdan

Assignment 5: Goal

Implement the wave equation using:

1. Implicit time-stepping
2. Spectral time-stepping

Implicit Time-Stepping: Where and What

`WaveEquation/WaveEquation.cpp`:

The executable solves the wave equation over a mesh using implicit time-stepping, visualized by offsetting the mesh in the normal direction, proportionally to the displacement value.

Displacements are specified by selecting a vertex at which to add/subtract displacement, or by enabling “rain”.

Implicit Time-Stepping: Where and What

`WaveEquation/WaveEquation.cpp`:

In the constructor:

WaveEquationViewer::WaveEquationViewer(…)

Implement the functionality initializing the solver member data:

WaveEquationViewer::_solver

using:

WaveEquationViewer::_M: The mass matrix associated with scalar functions

WaveEquationViewer::_S: The stiffness matrix associated with scalar functions

WaveEquationViewer::stepSize: The step-size

The matrices will have been initialized for you.

Implicit Time-Stepping: Where and What

`WaveEquation/WaveEquation.cpp`:

In the member function:

`WaveEquationViewer::_updateNumericalFactorization( … )`

Implement the functionality to update the solver when the user changes the step-size.

Recall that `Eigen`'s simplicial factorization has a member function `SimplicialLDLt::factorize` which only updates the numerical factorization.

Since the sparsity structure of the system matrix hasn't changed, there is no need to expend efforts recomputing the symbolic factorization.

Implicit Time-Stepping: Where and What

`WaveEquation/WaveEquation.cpp`:

In the member function:

WaveEquationViewer::animate(…)

Advance the displacements to the next time-step.

The displacements for the previous time-step will be stored in:

WaveEquationViewer::_mesh.values

When the current displacements are computed, you will over-write the previous displacement values.

Implicit Time-Stepping: Where and What

`WaveEquation/WaveEquation.cpp`:

In the member function:

Solving for the displacements at the current time-steps requires knowing the displacements from the previous time-steps **and** the displacements from the time-step before that (or the velocity, or the difference).

Add member data as necessary (and initialize in the constructor) for tracking previous displacement information.

Th

`WaveEquationViewer::_mesh.values`

When the current displacements are computed, you will over-write the previous displacement values.

Implicit Time-Stepping: Where and What

The executable takes the following parameters:

--in <input geometry>:
The file-name of the mesh (mandatory)

--stepSize <step-size>:
The step-size to be used for time-stepping the PDE – default 0.001.

--rain <rain-drop frequency>:
The frequency, in terms of frame count, with which random “rain-drops” strike the surface – default is 0, corresponding to no “rain”.

Implicit Time-Stepping: Where and What

You can interface with the running executable using:

[left click]: Add a negative displacement under the mouse

[right click]: Add a positive displacement under the mouse

‘+‘: Advance a single time-step

[space bar]: Start/stop the animation by hitting.

(Animation is on by default.)

‘[‘: Decrease step-size

‘]’: Increase step-size

‘r’: Specify rain-drop frequency

Spectral Time-Stepping: Where and What

`SpectralWaveEquation/SpectralWaveEquation.cpp`:

The executable solving the wave equation over a mesh using spectral time-stepping, visualized by offsetting the mesh in the normal direction in proportion to the displacement value.

Displacements are specified by selecting a vertex at which to add/subtract displacement, or by enabling “rain”.

[WARNING] The spectral decomposition is expensive (and will still only reproduce low-frequency signal)

Spectral Time-Stepping: Where and What

`SpectralWaveEquation/SpectralWaveEquation.cpp`:

The initial conditions for the PDE are stored in the member data:

SpectralWaveEquationViewer::_cosCoeff
SpectralWaveEquationViewer::_sinCoeff

You will need to:

- Convert from values at vertices to Fourier coefficients and back
- Compute the Fourier coefficients at time t
- Introduce offsets into the initial condition coefficients

Assume that $b_1 = 0$ in the term:

$$b_1 \cdot t \cdot \psi_1(p)$$

(i.e. that the average velocity is zero)

Spectral Time-Stepping: Where and What

`SpectralWaveEquation/SpectralWaveEquation.cpp`:

The solution to the (generalized) stiffness eigen-problem is stored in a templated object of type:

```
< typename SolverType > Spectrum::Spectrum< SolverType >
```

The template parameter `SolverType` specifies the type of solver to be used when implementing shift-and-invert.

Spectral Time-Stepping: Where and What

`SpectralWaveEquation/SpectralWaveEquation.cpp`:

The solution to the (generalized) stiffness eigen-problem is stored in a templated object of type:

```
< typename SolverType > Spectrum::Spectrum< SolverType >
```

The object supports getting the spatial and frequency dimensions:

```
unsigned int Spectrum< SolverType >::sDimension( void ) const
unsigned int Spectrum< SolverType >::fDimension( void ) const
```

The object supports getting the i-th eigenvalue and (unit-norm) eigenvector:

```
const double & eValue( unsigned int idx ) const
const Eigen::VectorXd & eVector( unsigned int idx ) const
```

Spectral Time-Stepping: Where and What

`SpectralWaveEquation/SpectralWaveEquation.cpp`:

The solution to the (generalized) stiffness eigen-problem is stored in a templated object of type:

`< typename SolverType > Spectrum::Spectrum< SolverType >`

The member data:

`SpectralWaveEquationViewer::_spectrum`

is initialized for you in the constructor:

`SpectralWaveEquationViewer::SpectralWaveEquationViewer`

`unsigned int Spectrum< SolverType >::Dimension( void ) const`

The object supports getting the i-th eigenvalue and (unit-norm) eigenvector:

`const double & eValue( unsigned int idx ) const`

`const Eigen::VectorXd & eVector( unsigned int idx ) const`

Spectral Time-Stepping: Where and What

`SpectralWaveEquation/SpectralWaveEquation.cpp`:

Compute the Fourier coefficients of a signal (values at vertices):

`Eigen::VectorXd ForwardFourier( ... )`

This function takes as input:

- `const Spectrum::Spectrum< SolverType > & spectrum`
  The eigenvalues/vectors of the (generalized) stiffness eigen-problem
- `const Eigen::SparseMatrix< double > & M`
  The mass matrix (for computing the inner-product of the signal with the eigen-functions)
- `const Eigen::VectorXd & x`
  The signal whose coefficients are to be computed – of size `spectrum.sDimension()`

The function returns:

- The Fourier coefficients of the input signal – of size `spectrum.fDimension()`

Spectral Time-Stepping: Where and What

`SpectralWaveEquation/SpectralWaveEquation.cpp`:

Compute the Fourier coefficients of a signal:

`Eigen::VectorXd ForwardFourier( … )`

Compute the signal represented by Fourier coefficients:

`Eigen::VectorXd InverseFourier( … )`

This function takes as input:

- `const Spectrum::Spectrum< SolverType > & spectrum`
  The eigenvalues/vectors of the (generalized) stiffness eigen-problem
- `const Eigen::VectorXd & f`
  The fourier coefficients of the signal – of size `spectrum.fDimension()`

The function returns:

- The signal represented by the Fourier coefficients – of size `spectrum.sDimension()`

Spectral Time-Stepping: Where and What

`SpectralWaveEquation/SpectralWaveEquation.cpp`:

Compute the Fourier coefficients of a signal:

`Eigen::VectorXd ForwardFourier( … )`

Compute the signal represented by Fourier coefficients:

`Eigen::VectorXd InverseFourier( … )`

This function takes as input:

The code defines a function:

`SanityCheckFourier(…)`

that confirms that the `ForwardFourier` and `InverseFourier` functions are consistently implemented. It is invoked in the constructor:

`SpectralWaveEquationViewer::SpectralWaveEquationViewer`

- The signal represented by the Fourier coefficients – of size `spectrum.sDimension()`

Spectral Time-Stepping: Where and What

`SpectralWaveEquation/SpectralWaveEquation.cpp`:

Compute the Fourier coefficients of the displacement's value at $t = t_0$:

`Eigen::VectorXd GetWaveValueCoefficients(...)`

This function takes as input:

- `const Spectrum::Spectrum< SolverType > & spectrum`
  The eigenvalues/vectors of the (generalized) stiffness eigen-problem
- `const Eigen::VectorXd & cosCoeff`, `const Eigen::VectorXd & sinCoeff`
  The coefficients describing the initial conditions – both of size `spectrum.fDimension()`
- `double time`
  The current time

The function returns:

- The Fourier coefficients of the displacement's value – of size `spectrum.fDimension()`

Spectral Time-Stepping: Where and What

`SpectralWaveEquation/SpectralWaveEquation.cpp`:

Compute the Fourier coefficients of the displacement's velocity at $t = t_0$:

`Eigen::VectorXd GetWaveVelocityCoefficients(...)`

This function takes as input:

- `const Spectrum::Spectrum< SolverType > & spectrum`
  The eigenvalues/vectors of the (generalized) stiffness eigen-problem
- `const Eigen::VectorXd & cosCoeff`, `const Eigen::VectorXd & sinCoeff`
  The coefficients describing the initial conditions – both of size `spectrum.fDimension()`
- `double time`
  The current time

The function returns:

- The Fourier coefficients of the displacement's velocity – of size `spectrum.fDimension()`

Spectral Time-Stepping: Where and What

`SpectralWaveEquation/SpectralWaveEquation.cpp`:

Temporally shift initial conditions, so they describe the displacement's value and velocity at time $t = t_0$.

Motivation:

Given a user interaction at time $t = t_0$, we define an offset that we would like to apply to the current displacement, $\delta: \mathcal{M} \to \mathbb{R}$.*

Since we represent displacement in terms of the **Fourier coefficients** of the values and velocity **at time $\boldsymbol{t = 0}$**, to incorporate the offset, we need to:

- Compute the Fourier coefficients of the offset's value at time $t = t_0$:

$$\delta(p) = \widehat{\boldsymbol{\delta}}_1 \cdot \psi_1(p) + \cdots + \hat{\delta}_{|\mathcal{V}|} \cdot \psi_{|\mathcal{V}|}(p)$$

- Compute the Fourier coefficients of the offset's value and velocity that we would have to have at time $t = 0$ so that evaluated at time $t = t_0$ it gives offset value δ.
- Add those Fourier coefficients into the displacement.

*Assume the offset velocity is zero.

Spectral Time-Stepping: Where and What

`SpectralWaveEquation/SpectralWaveEquation.cpp`:

Temporally shift initial conditions, so they describe the displacement's value and velocity at time $t = t_0$.

Formally:

Given a trigonometric function expressed as:

$$f(t) = a_0 \cdot \cos(\alpha \cdot t) + b_0 \cdot \sin(\alpha \cdot t)$$

the temporally shifted function will be:

$$\begin{aligned} g(t) &\equiv f(t - t_0) \\ &= a_0 \cdot \cos(\alpha \cdot t - \alpha \cdot t_0) + \sin(\alpha \cdot t - \alpha \cdot t_0) \\ &= a_0 \cdot (\cos(\alpha \cdot t) \cdot \cos(\alpha \cdot t_0) + \sin(\alpha \cdot t) \cdot \sin(\alpha \cdot t_0)) \\ &- b_0 \cdot (\cos(\alpha \cdot t) \cdot \sin(\alpha \cdot t_0) + \sin(\alpha \cdot t) \cdot \cos(\alpha \cdot t_0)) \\ &= \cos(\alpha \cdot t) \cdot (a_0 \cdot \cos(\alpha \cdot t_0) - b_0 \cdot \sin(\alpha \cdot t_0)) \\ &+ \sin(\alpha \cdot t) \cdot (a_0 \cdot \sin(\alpha \cdot t_0) + b_0 \cdot \cos(\alpha \cdot t_0)) \end{aligned}$$

Spectral Time-Stepping: Where and What

`SpectralWaveEquation/SpectralWaveEquation.cpp`:

Temporally shift initial conditions, so they describe the displacement's value and velocity at time $t = t_0$:

`void ShiftWaveCoefficients( … )`

This function takes as input:

- `const Spectrum::Spectrum< SolverType > & spectrum`
  The eigenvalues/vectors of the (generalized) stiffness eigen-problem
- `Eigen::VectorXd & cosCoeff`, `Eigen::VectorXd & sinCoeff`
  The coefficients describing the initial conditions – both of size `spectrum.fDimension()`
  These will be updated to hold the shifted coefficients
- `double time`
  The time to shift to

Spectral Time-Stepping: Where and What

The executable takes the following parameters:

--in <input geometry>:
The file-name of the mesh (mandatory)

--stepSize <step-size>:
The step-size to be used for time-stepping the PDE – default is 0.001.

--sDimension <spectral dimension>:
The number of (lower) eigenvectors/values to compute – default is 400.

--rain <rain-drop frequency>:
The frequency, in terms of frame count, with which random “rain-drops” strike the surface – default is 0, corresponding to no “rain”.

Spectral Time-Stepping: Where and What

You can interface with the running executable using:

[left click]: Add a negative displacement under the mouse

[right click]: Add a positive displacement under the mouse

‘+‘: Advance a single time-step

[space bar]: Start/stop the animation by hitting.

(Animation is on by default.)

‘[‘: Decrease step-size

‘]’: Increase step-size

‘r’: Specify rain-drop frequency

Assignment 5: Questions

WaveEquation:

- Given an initial displacement, what happens over time?

SpectralWaveEquation:

- Given an initial displacement, what happens over time?
- How does quality and performance change with increased spectral dimension?