Outline

- Halfedge data structure
- Sparse matrices
- Solving linear systems (direct methods)
- Intro to either C++ or JS
The Halfedge Data Structure
The Halfedge Data Structure

```c
struct Halfedge
{
    Halfedge twin;
    Halfedge next;
    Vertex vertex;
    Edge edge;
    Face face;
};

struct Edge
{
    Halfedge halfedge;
};

struct Face
{
    Halfedge halfedge;
};

struct Vertex
{
    Halfedge halfedge;
};
```
The Halfedge Data Structure

How would I find the faces adjacent to an edge?

Given: Edge e

e.halfedge
The Halfedge Data Structure

How would I find the faces adjacent to an edge?

Given: `Edge e`

```
Halfedge he = e.halfedge;
Face left_face = he.face;
Face right_face = he.twin.face;
```
The Halfedge Data Structure

How would I find the edges adjacent to a triangle?

Given: \textit{Face} \ tri

\texttt{tri.halfedge}
The Halfedge Data Structure

How would I find the edges adjacent to a triangle?

Given: Face tri

```
Halfedge he = tri.halfedge;
Edge e1 = he.edge;
Edge e2 = he.next.edge;
Edge e3 = he.next.next.edge;
```
The Halfedge Data Structure

How would I loop over the edges adjacent to a polygon?

Given: Face \( f \)

\[
\text{struct Halfedge} \\
\{ \\
\quad \text{Halfedge twin;} \\
\quad \text{Halfedge next;} \\
\quad \text{Vertex vertex;} \\
\quad \text{Edge edge;} \\
\quad \text{Face face;} \\
\};
\]

\[
\text{struct Edge} \\
\{ \\
\quad \text{Halfedge halfedge;} \\
\};
\]

\[
\text{struct Vertex} \\
\{ \\
\quad \text{Halfedge halfedge;} \\
\};
\]
The Halfedge Data Structure

How would I loop over the edges adjacent to a polygon?

Given: `Face f`

```c
Halfedge start = f.halfedge;
Halfedge he = start;
do {
    Edge e = he.edge;
    /* Some code */
    he = he.next;
} while (he != start);
```
The Halfedge Data Structure

How would I loop over the edges adjacent to a vertex?

Given: $\text{Vertex } v$

$\text{v.halfedge}$
The Halfedge Data Structure

How would I loop over the edges adjacent to a vertex?

Given: `Vertex v`

```c
Halfedge start = v.halfedge;
Halfedge he = start;
do {
    Edge e = he.edge;
    /* Some code */
    he = he.twin.next;
} while (he != start);
```
Many convenience functions in both JS and C++!

- **f.adjacentVertices()** → iterator over vertices adjacent to face \( f \)
- **v.adjacentVertices()** → iterator over vertices adjacent to vertex \( v \)
- **v.adjacentHalfedges()**
- **v.outgoingHalfedges()** → iterator over halfedges whose tail is vertex \( v \)

... etc.

See individual documentation for library-specific usage
Storing Matrices
Matrices

How can I write down a matrix?

• Option 2: 2D array

• If your matrix doesn’t have much structure, this might be the best you can do

• But it can take a lot of space to write down an entire matrix

• And working with (really) big matrices is slow
Matrices

- What matrices do we care about?

- It turns out that *adjacency matrices* are very important.

\[
E^0 = \begin{bmatrix}
0 & 1 & 2 & 3 \\
1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 \\
\end{bmatrix}, \quad E^1 = \begin{bmatrix}
0 & 1 & 2 & 3 & 4 & 5 \\
0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 \\
2 & 0 & 1 & 1 & 0 & 0 \\
2 & 0 & 0 & 0 & 1 & 1 \\
3 & 0 & 0 & 0 & 1 & 1 \\
\end{bmatrix}
\]
Matrices

- Most entries are 0!
- We can improve our lives by only storing nonzero entries → sparse matrices
Aside: Sparse Matrix Formats

• Important format: Compressed Sparse Row (CSR)
• Store the nonzero entries in row-major order, and some information about spacing
• Row-major order => matrix-vector products are fast

\[
\begin{align*}
A[i] &= \text{entries} \\
IA[i] &= \text{total number of nonzero entries before row } i \\
JA[i] &= \text{column of the } i\text{th entry of } A
\end{align*}
\]
Aside: Sparse Matrix Formats

$A[i] =$ entries
$IA[i] =$ total number of nonzero entries before row $i$
$JA[i] =$ column of the $i$th entry of $A$

$A = \begin{pmatrix} 5 & 8 & 3 & 6 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 6 & 0 & 0 \end{pmatrix}$

$IA = \begin{pmatrix} 0 & 0 & 2 & 3 & 4 \end{pmatrix}$

$JA = \begin{pmatrix} 0 & 1 & 2 & 1 \end{pmatrix}$
Aside: Sparse Matrix Formats

- There’s also Compressed Sparse Column (CSC)
- Fast to multiply CSC by row vectors
- Both are slow to add elements to
  - Usually you build the matrix in another format, then convert before doing computation
Solving Linear Systems
Linear Systems of Equations

Linear algebra review

\[
\begin{align*}
  x + 2y - 4z &= 1 \\
  3x - 5y + 7z &= 2 \\
  -x + 3y + 5z &= -2
\end{align*}
\]

\[
\begin{pmatrix}
  1 & 2 & -4 \\
  3 & -5 & 7 \\
  -1 & 3 & 5
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix}
= 
\begin{pmatrix}
  1 \\
  2 \\
  -2
\end{pmatrix}
\]

\[Ax = b\]
Linear Systems of Equations

- How do we solve $Ax = b$?
- Compute the inverse / Gaussian Elimination
- Not good for sparse matrices
Linear Systems of Equations

• Some special cases are easy

• What if A is diagonal?

\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & -3
\end{pmatrix}
\begin{pmatrix}
x \\ y \\ z
\end{pmatrix}
=
\begin{pmatrix}
1 \\ 4 \\ 6
\end{pmatrix}
\]
Linear Systems of Equations

• What if $A$ is lower-triangular?

\[
\begin{pmatrix}
1 & 0 & 0 \\
1 & 2 & 0 \\
2 & 3 & -3
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z
\end{pmatrix}
=
\begin{pmatrix}
1 \\
5 \\
11
\end{pmatrix}
\]

- $x = 1$
- $x + 2y = 5 \Rightarrow y = 2$
- $2x + 3y - 3z = 11 \Rightarrow z = -1$

• (Same trick works if $A$ is upper-triangular)
Linear Systems of Equations

• Can this help us with arbitrary linear systems?

• Yes!

• Given an invertible matrix $A$, we can factor it as a lower-triangular matrix times an upper triangular matrix:

$$A = LU$$

$$\begin{pmatrix} 4 & 3 \\ 6 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1.5 & 1 \end{pmatrix} \begin{pmatrix} 4 & 3 \\ 0 & -1.5 \end{pmatrix}$$
LU Decomposition

\[ Ax = b \]

\[ LUx = b \]

\[ Ly = b \text{ and } y = Ux \]
LU Decomposition

• How do we compute LU decomposition?
• Simple solution - run Gaussian Elimination half way
  • Problem - still not good for sparse matrices
• We’ll use a fancier implementation
Cholesky Decomposition

• If $A$ is symmetric and positive-semidefinite, then the LU decomposition is really nice

\[ A = LL^T \]

• Called Cholesky or $LL^T$ decomposition
QR Decomposition

- LU and Cholesky decompositions take advantage of the fact that it’s easy to solve triangular systems.
- It’s also easy to solve systems given by rotation matrices.

\[
Q^{-1} = Q^T \\
Qx = b \Rightarrow x = Q^Tb
\]
QR Decomposition

• Any square matrix can be decomposed as QR for Q a rotation and R upper triangular

• There are also versions for rectangular matrices

\[
Ax = b
\]

\[
QRx = b
\]

\[
Qy = b \text{ and } y = Rx
\]
QR Decomposition

- Also available in framework
- Not as fast as Cholesky but more widely applicable
ddg-exercises-js
ddg-exercises-js

- Repository on Github
- [https://github.com/cmu-geometry/ddg-exercises-js](https://github.com/cmu-geometry/ddg-exercises-js)
- Contains all assignments for the semester
Javascript

• Feels similar to C, C++, Java, …. Really any language with braces

• Runs in your browser, so there isn’t too much setup

• You probably won’t need to use any fancy features particular to Javascript - just need some functions, conditionals, loops, etc
ddg-exercises-js

• Documentation included
ddg-exercises-js/docs/index.html

• Coding assignments
ddg-exercises-js/projects

• Tests
ddg-exercises-js/tests

ddg-exercises-js is a fast and flexible framework for 3D geometry processing on the web! Easy integration with HTML, WebGL, makes it particularly suitable for things like mobile apps, online demos, and course content. For many tasks, performance comes within striking distance of native (C++) code. Plus, since the framework is pure JavaScript, no compilation or installation is necessary on any platform. Moreover, geometry processing algorithms can be edited in the browser (using for instance the JavaScript Console in Chrome).

At a high level, the framework is divided into three parts - an implementation of a halfedge mesh data structure, an optimized linear algebra package and skeleton code for various geometry processing algorithms. Each algorithm comes with its own viewer for rendering.

Detailed documentation and unit tests for each of these parts can be found in the docs and tests directories of this repository.

Getting started

1. Clone the repository and change into the projects directory

```
$ git clone https://github.com/cmu-geometry/ddg-exercises-js.git
$ cd ddg-exercises-js/projects
```

2. Open the index.html file in any of the sub directories in a browser of your choice (Chrome and Firefox usually provide better rendering performance than Safari).

Dependencies (all included)

1. Linear Algebra - A wrapper around the C++ library Eigen compiled to asm.js with emscripten. Future updates will compile the more optimized sparse matrix library Suitesparse to asm.js.
2. Rendering - three.js
3. Unit Tests - Mocha and Chai

About Javascript

The implementation of ddg-exercises-js attempts to minimize the use of obscure Javascript language features. It should not be too difficult for anyone with experience in modern web development to learn. Additionally, you can run the code in the browser console to check all the features in action!
Documentation

Class: Mesh

Core. Mesh

new Mesh()
This class represents a Mesh.

Properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertices</td>
<td>Array.&lt;module.Core.Vertex&gt;</td>
<td>The vertices of the mesh</td>
</tr>
<tr>
<td>edges</td>
<td>Array.&lt;module.Core.Edge&gt;</td>
<td>The edges of the mesh</td>
</tr>
<tr>
<td>faces</td>
<td>Array.&lt;module.Core.Face&gt;</td>
<td>The faces of the mesh</td>
</tr>
<tr>
<td>corners</td>
<td>Array.&lt;module.Core.Corners&gt;</td>
<td>The corners of the mesh</td>
</tr>
<tr>
<td>halfedges</td>
<td>Array.&lt;module.Core.Halfedge&gt;</td>
<td>The halfedges of the mesh</td>
</tr>
<tr>
<td>boundaries</td>
<td>Array.&lt;module.Core.Boundary&gt;</td>
<td>The boundaries of the mesh</td>
</tr>
<tr>
<td>generators</td>
<td>Array.&lt;Array.</td>
<td>An array of arrays representing homology generators</td>
</tr>
</tbody>
</table>

Methods

<static> fromTriplet(T)
Initializes a sparse matrix from a Triplet object.

Parameters:
- Name: Type: Description:

Class: SparseMatrix

LinearAlgebra. SparseMatrix

new SparseMatrix()
This class represents a m by n real matrix where only nonzero entries are stored explicitly. Do not create a SparseMatrix from its constructor, instead use static factory methods such as fromTriplet, identity and diag.

Example

```javascript
let T = new Triplet(100, 100);
T.addEntry(33, 1, 43);
T.addEntry(44, 99, 99);
let A = SparseMatrix.fromTriplet(T);
```

Methods

Methods

ddg-exercises-js

ddg-exercises-js is a fast and flexible framework for 3D geometry processing on the web! Easy integration with HTML5/WebGL makes it particularly suitable for things like mobile apps, online demos, and course content. For many tasks, performance comes within striking distance of native (C++) code. Plus, since the framework is pure JavaScript, no compilation or installation is necessary on any platform. Moreover, geometry processing algorithms can be edited in the browser (using for instance the JavaScript Console in Chrome).

At a high level, the framework is divided into three parts - an implementation of a halfedge mesh data structure, an optimized linear algebra package and skeleton code for various geometry processing algorithms. Each algorithm comes with its own viewer for rendering.

Detailed documentation and unit tests for each of these parts can be found in the docs and tests directories of this repository.

Getting started

1. Clone the repository and change into the projects directory
2. Open the index.html file in any of the sub-directories in a browser of your choice (Chrome and Firefox usually provide better rendering performance than Safari).

Dependencies (all included)

1. Linear Algebra - A wrapper around the C++ library Eigen compiled to asm.js with emscripten. Future updates will compile to WebAssembly.
Coding Assignments

- Viewers

  ddg-exercises-js/projects/simplicial-complex-operators/index.html

- Write code in project folder or one of the modules
- Graphics programming often involves a lot of boilerplate before getting started drawing - We’ve mostly done that for you. You just have to fill in the interesting bits
Tests

• Test scripts

ddg-exercises-js/tests/simplicial-complex-operators/test.html

• As you write your code, you should see it pass more tests
Navigating halfedges
In ddg-exercises-js

Class: Mesh
Core. Mesh

new Mesh()
This class represents a Mesh.

Properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertices</td>
<td>Array<a href="">module:Core:Vertex</a></td>
<td>The vertices contained in this mesh.</td>
</tr>
<tr>
<td>edges</td>
<td>Array<a href="">module:Core:Edge</a></td>
<td>The edges contained in this mesh.</td>
</tr>
<tr>
<td>faces</td>
<td>Array<a href="">module:Core:Face</a></td>
<td>The faces contained in this mesh.</td>
</tr>
<tr>
<td>corners</td>
<td>Array<a href="">module:Core:Corner</a></td>
<td>The corners contained in this mesh.</td>
</tr>
<tr>
<td>halfedges</td>
<td>Array<a href="">module:Core:Halfedge</a></td>
<td>The halfedges contained in this mesh.</td>
</tr>
<tr>
<td>boundaries</td>
<td>Array<a href="">module:Core:Face</a></td>
<td>The boundary loops contained in this mesh.</td>
</tr>
</tbody>
</table>

| generators | Array<Array.<module:Core:Halfedge>> | An array of halfedge arrays, i.e., | | \( h_{11}, h_{21}, ..., h_{1n}, h_{12}, h_{22}, ..., h_{nm} \) | representing this mesh's homology generators. |

Methods
In ddg-exercises-js

Includes many convenience functions

adjacentVertices()
Convenience function to iterate over the vertices in this face. Iterates over the vertices of a boundary loop if this face is a boundary loop.

Returns:
Type
module:Core.Vertex

Example

```javascript
let f = mesh.faces[0]; // or let b = mesh.boundaries[0]
for (let v of f.adjacentVertices()) {
  // Do something with v
}
```

adjacentEdges()
Convenience function to iterate over the edges adjacent to this vertex.

Returns:
Type
module:Core.Edge

Example

```javascript
let v = mesh.vertices[0];
for (let e of v.adjacentEdges()) {
  // Do something with e
}
```
Linear algebra in ddg-exercises-js
Sparse Matrices in ddg-exercises-js

**Class: SparseMatrix**

LinearAlgebra. SparseMatrix

```javascript
new SparseMatrix()
```

This class represents an m by n real matrix where only nonzero entries are stored explicitly. Do not create a SparseMatrix from its constructor; instead use static factory methods such as fromTriplet, identity and diag.

Example

```javascript
let T = new Triplet(100, 100);
T.addEntry(1, 1, 45);
T.addEntry(0, 0, 99);
let A = SparseMatrix.fromTriplet(T);
let B = SparseMatrix.identity(10, 10);
let d = DenseMatrix.ones(100, 1);
let C = SparseMatrix.diag(d);
```

**Methods**

<static> fromTriplet(T)

Initializes a sparse matrix from a Triplet object.

**Class: Triplet**

LinearAlgebra. Triplet

```javascript
new Triplet(m, n)
```

This class represents a small structure to hold nonzero entries in a SparseMatrix. Each entry is a triplet of a value and the (i, j)th indices, i.e., (v, i, j).

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>number</td>
<td>The number of rows in the sparse matrix that will be initialized from this triplet.</td>
</tr>
<tr>
<td>n</td>
<td>number</td>
<td>The number of columns in the sparse matrix that will be initialized from this triplet.</td>
</tr>
</tbody>
</table>
Warning

• How do you represent a vector?
• LinearAlgebra.Vector only represents 3D vectors
• Instead, construct a matrix with \( n \) rows and 1 column
• Multiply matrices by vectors using `timesDense` or `timesSparse`
Solving linear systems

Class: Cholesky

LinearAlgebra. Cholesky

new Cholesky()

This class represents a Cholesky L*L^T factorization of a square SparseMatrix. The factorization is computed on the first call to solve and is reused in subsequent calls to solvePositiveDefinite (e.g., right-hand side b of the linear system Ax = b changes) unless the sparse matrix itself is altered through operations such as *, +=, and -=. Do not use the constructor to initialize a sparse matrix.

Methods

solvePositiveDefinite()

Solves the linear system Ax = b, where A is a square positive definite matrix.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>module:LinearAlgebra.DenseMatrix</td>
<td>The dense right-hand side b.</td>
</tr>
</tbody>
</table>

Chol()

Returns a sparse Cholesky factorization of this sparse matrix.

Returns:

Type module:LinearAlgebra.Cholesky

Class: LU

LinearAlgebra. LU

new LU()

This class represents a LU factorization of a rectangular SparseMatrix. The factorization is computed on the first call to solve or solveSquare (e.g., when only the right-hand side b of the linear system Ax = b changes) unless the sparse matrix itself is altered through operations such as *, +=, and -=. Do not use the constructor to initialize a sparse matrix.

lu()

Returns a sparse LU factorization of this sparse matrix.

Returns:

Type module:LinearAlgebra.LU

Class: QR

LinearAlgebra. QR

new QR()

This class represents a QR factorization of a rectangular SparseMatrix. The factorization is computed on the first call to solve, and is reused in subsequent calls to solve (e.g., when only the right-hand side b of the linear system Ax = b changes) unless the sparse matrix itself is altered through operations such as *, +=, and -=. Do not use the constructor to initialize a sparse matrix directly.

qr()

Returns a sparse QR factorization of this sparse matrix.

Returns:

Type module:LinearAlgebra.QR
Print statements

Print using `console.log()`

Console is usually under “Developer tools” - might be different in your browser
ddg-exercises (C++)
Welcome to Geometry Central

Geometry-central is a modern C++ library of data structures and algorithms for geometry processing, with a particular focus on surface meshes.

Features include:

- A polished surface mesh class, with efficient support for the system of containers for associating data with mesh element
- Implementations of canonical geometric quantities on surface normals and curvatures to tangent vector bases to operators differential geometry.
- A suite of powerful algorithms, including computing distance generating direction fields, and manipulating intrinsic: Delaunay
- A coherent set of sparse linear algebra tools, based on Eigen automatically utilize better solvers if available on your system

Polyscope is a C++/Python viewer and user interface for 3D data, like meshes and point clouds. Scientists, engineers, artists, and hackers can use Polyscope to prototype algorithms—It is designed to easily integrate with existing codebases and popular libraries. The lofty objective of Polyscope is to offer a useful visual interface to your data via a single line of code.

Polyscope uses a paradigm of structures and quantities. A structure is a geometric object in the scene, such as a surface mesh or point cloud. A quantity is data associated with a structure, such as a scalar function or a vector field.

When any of these structures and quantities are registered, Polyscope displays them in an interactive 3D scene, handling boilerplate concerns such as toggling visibility, color-mapping data and adjusting maps, "picking" to click in the scene and query numerical quantities, etc.
ddg-exercises

- Repository on Github: https://github.com/GeometryCollective/ddg-exercises
- Clone recursively!
ddg-exercises

This repo contains C++ skeleton code for course assignments from Discrete Differential Geometry (15-458/858).

For the JavaScript version, see https://github.com/cmu-geometry/ddg-exercises-js.

This code framework uses Geometry Central for geometry processing utilities and Polyscope for visualization, which were developed by Nick Sharp and others in the Geometry Collective. Extensive documentation for these libraries --- and how to build them on various platforms --- can be found at the preceding links. If you're having trouble building, please make sure to take a look before bugging the TAs! :-( (We are of course still very happy to help if you're still having trouble.)

Documentation for Geometry Central can be found here.

Documentation for Polyscope can be found here.

Getting started

Clone the repository and its submodules.

git clone --recursive https://github.com/GeometryCollective/ddg-exercises
cd ddg-exercises/projects

Each project in ddg-exercises/projects builds its own executable when compiled. To run a particular project <project>, go to the projects/<project> directory. The basic process for compiling is as follows. First, make a build directory and compile using

mkdir build
cd build
make

This builds an executable main which can then be run using

bin/main <optional_path_to_a_mesh>

(See Geometry Central: Building for additional compiler flag options.)

• All coding assignments
  ddg-exercises/projects
• Additional READMEs per assignment
• Unit tests included
  - built in separate executable
Documentation

- Detailed documentation at https://geometry-central.net/!

- The sections most relevant to us are:
  - For vertex, edge, face objects, etc:
    Surface → Surface Mesh → Elements
  - For traversing the mesh:
    Surface → Surface Mesh → Navigation and Iteration
  - To get quantities associated with mesh elements (edge length, edge vector, face area, etc.):
    Geometry → Quantities
  - Sparse matrices:
    Numerical → Linear Algebra Utilities
  - Solving sparse linear systems:
    Numerical → Linear Solvers
Tests

- Tests are built along with everything else when you compile
- Run `bin/test-*`
- As you write your code, you should see it pass more tests
Assignments

- Write code in project folder or `core/`, in one or more of the source (.cpp) files
- We’ve handled visualization in Polyscope
- Generate Makefile using cmake
  - `make` and `bin/main` to run program!
- Additional meshes provided in `inputs/` (up a few directories relative to `projects/`
Navigating halfedges
In Geometry Central

**Halfedge**

A halfedge is a basic building block of a mesh. A halfedge is half of an edge, connecting two vertices on the same face. The halfedge is directed, from its start vertex to its end vertex, with a clockwise orientation: the halfedges with in an interior face will point in opposite directions. On a manifoldSurfaceMesh, all the halfedges (and an edge) will point in opposite directions.

**Traversal**:

- `Halfedge halfedge: twin()`
- `Halfedge halfedge: opposite()`
- `Halfedge halfedge: next()`
- `Halfedge halfedge: vertex()`
- `Halfedge halfedge: tailVertex()`
- `Halfedge halfedge: tipVertex()`
- `Edge halfedge: edge()`
- `Face halfedge: face()`
- `Corner halfedge: corner()`

**Vertex**

A vertex is a 0-dimensional point which serves as a node in the mesh.

**Traversal**:

- `Halfedge vertex: halfedge()`
- `Corner vertex: corner()`

**Edge**

An edge is a 1-dimensional element that connects two vertices in the mesh.

**Traversal**:

- `Halfedge edge: halfedge()`
- `Vertex edge: starterVertex/vertex X`
- `Vertex edge: finisherVertex/vertex Y`
- `Vertex edge: secondVertex/vertex Y`

**Face**

A face is a 2-dimensional element formed by a loop of 3 or more edges. In general, faces are planar, though many of the routines in geomcentral are only valid on triangular meshes.

**Traversal**:

- `Halfedge face: halfedge()`
- `BoundaryLoop face: halfedgeFace/face X`
In Geometry Central

Includes many convenience functions

(see Navigation and Iteration documentation)
Linear algebra in Geometry Central
Sparse Matrices in Geometry Central

- Geometry Central provides convenient functions for initialization
- G-C sparse matrices are Eigen matrices under the hood, so you can also initialize from Eigen sparse matrix

- Can also initialize from triplets, following [Eigen tutorial](#):
Solving linear systems

Direct solvers

These solvers provide a simple interface for solving sparse linear $Ax = b$.

A key feature is that these solvers abstract over the underlying numerical library. In their most basic form, Eigen’s sparse solvers will be used, and are always available. However, if present, the more-powerful Suitesparse solvers will be used instead. See the dependencies section for instruction to build with Suitesparse support.

As always, be sure to compile with optimizations for meaningful performance. In particular, Eigen’s built-in solvers will be very slow in debug mode (though the Eigen QR solver is always slow).

Quick solves

These are one-off routines for quick solves.

```cpp
Vector<T> solve(SparseMatrix<T> & matrix, const Vector<T> & rhs);
```

Solve a system with a square matrix. Uses an LU decomposition internally.

```cpp
Vector<T> solveSquare(SparseMatrix<T> & matrix, const Vector<T> & rhs);
```

```cpp
Vector<T> solvePositiveDefinite(SparseMatrix<T> & matrix, const Vector<T> & rhs);
```

Solve a system with a symmetric positive (semi-)definite matrix. Uses an LDLT decomposition internally.
Discrete Differential Geometry: An Applied Introduction

Thanks!