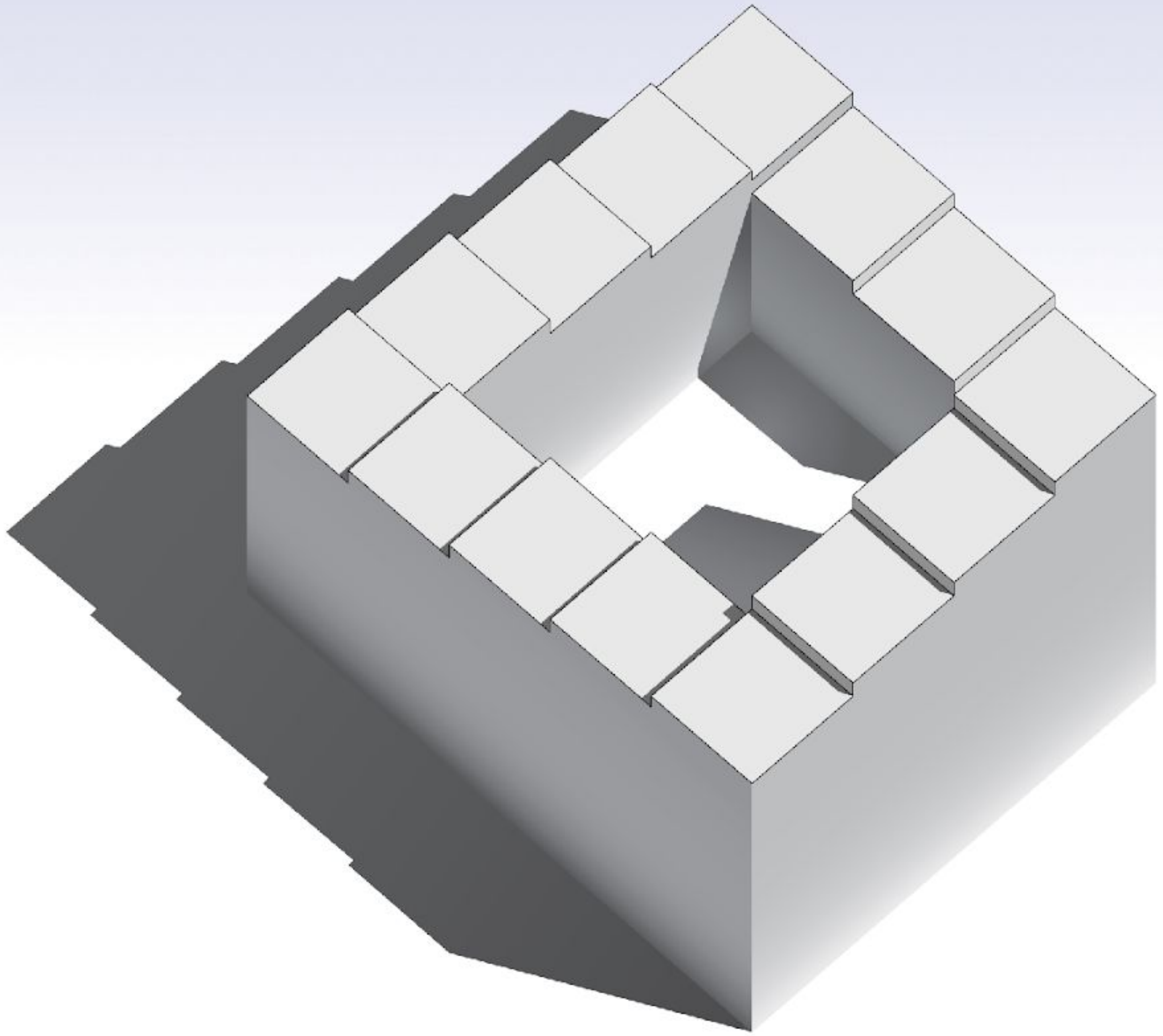
Recitation



Discrete Differential Geometry: An Applied Introduction

Spring 2024 TAs: Oscar Chen and Olga Guțan

Original slides by Mark Gillespie • Updated by Nicole Feng & Ethan Lu

Outline

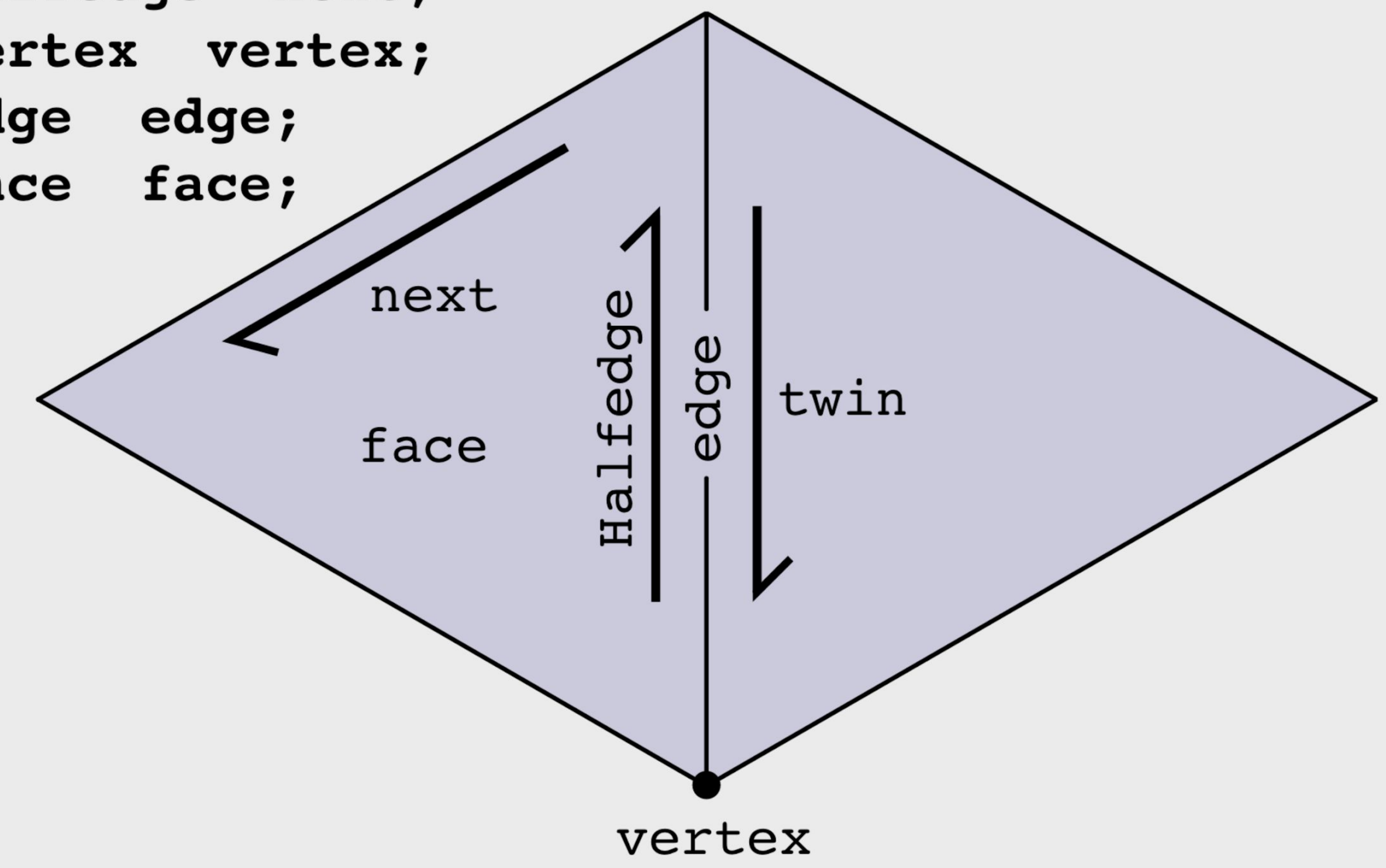
Outline

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

The Halfedge Data Structure

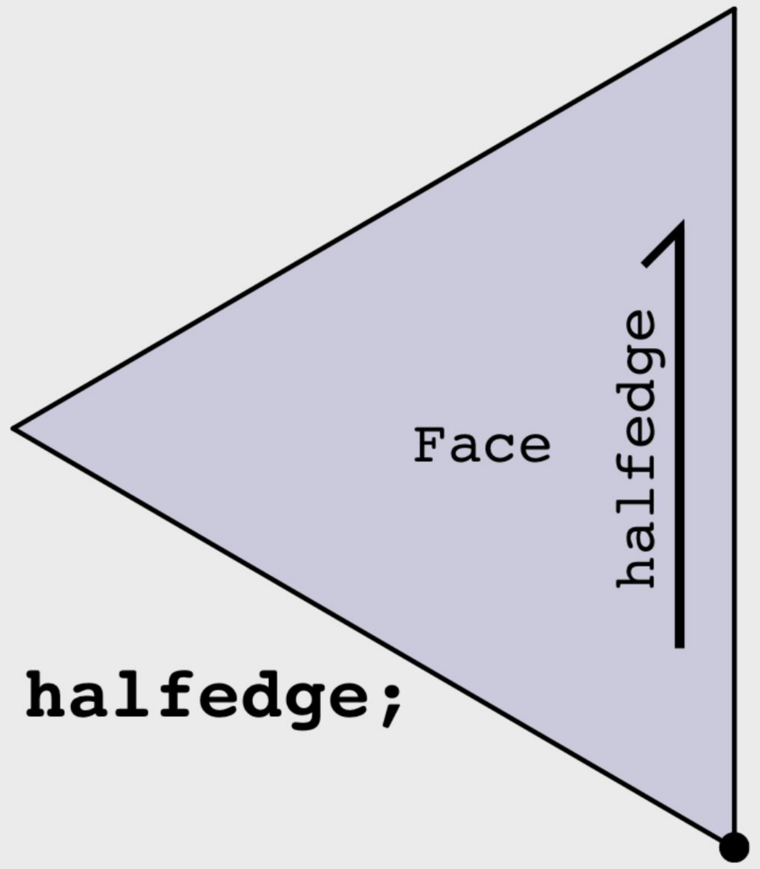
The Halfedge Data Structure

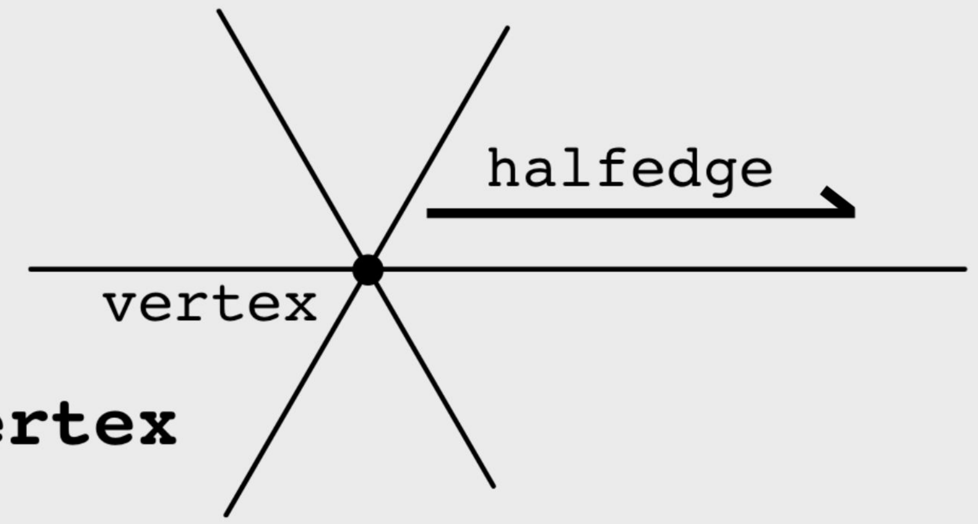
```
struct Halfedge
{
   Halfedge  twin;
   Halfedge  next;
   Vertex  vertex;
   Edge  edge;
   Face  face;
};
```



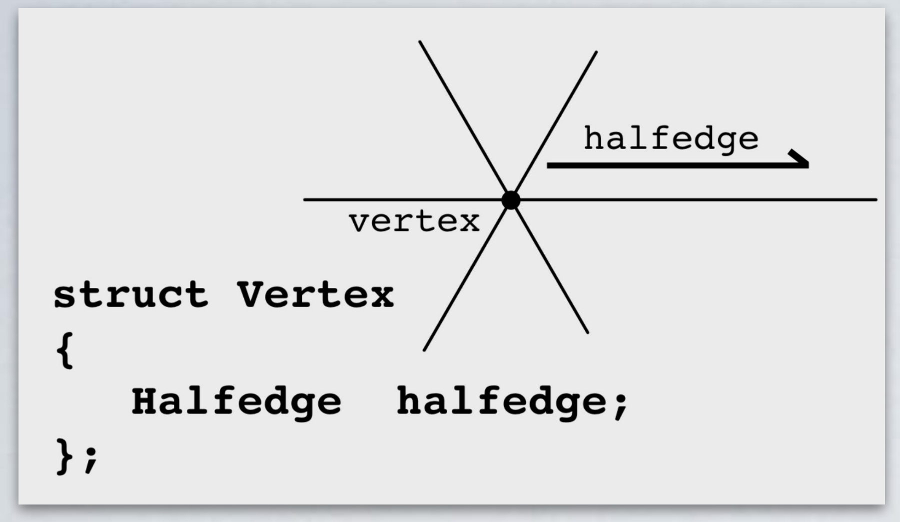

```
struct Edge
{
   Halfedge  halfedge;
};
```

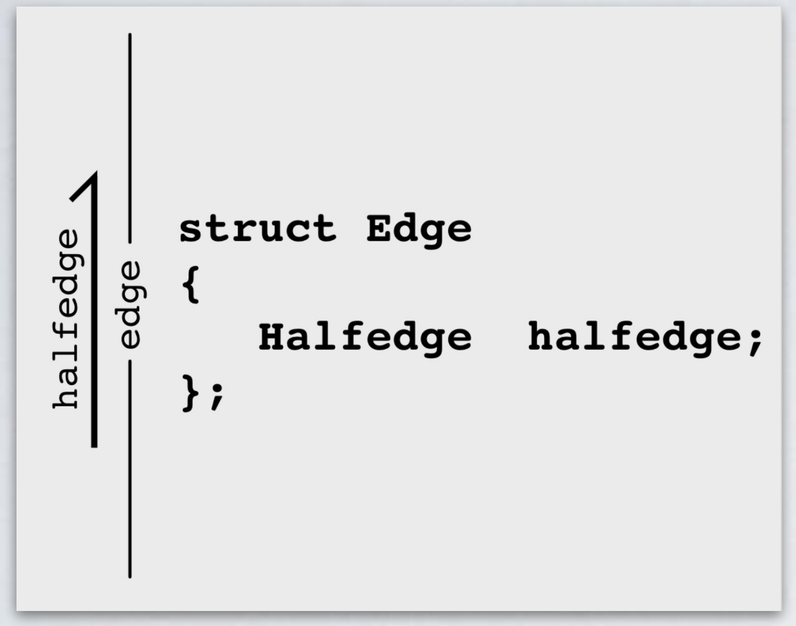
```
struct Face
{
   Halfedge  halfedge;
};
```

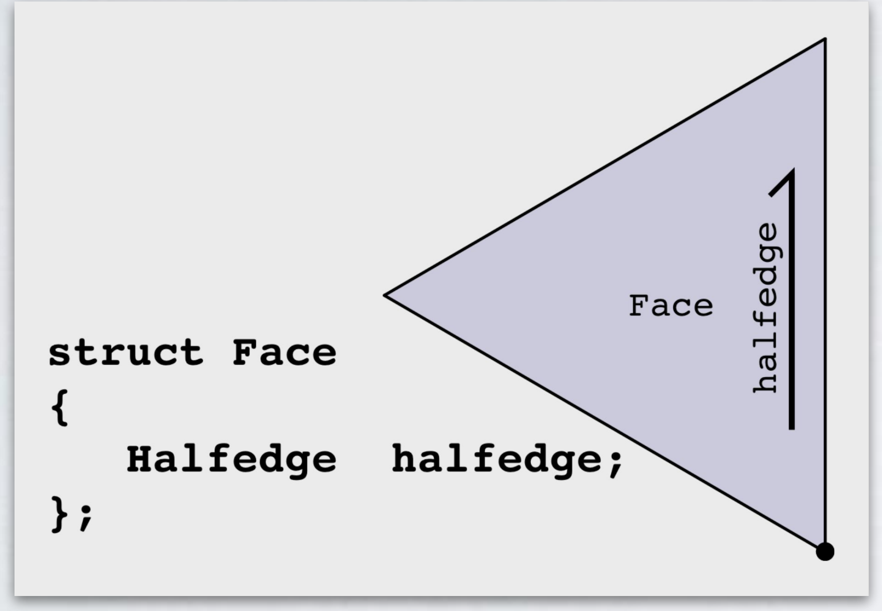


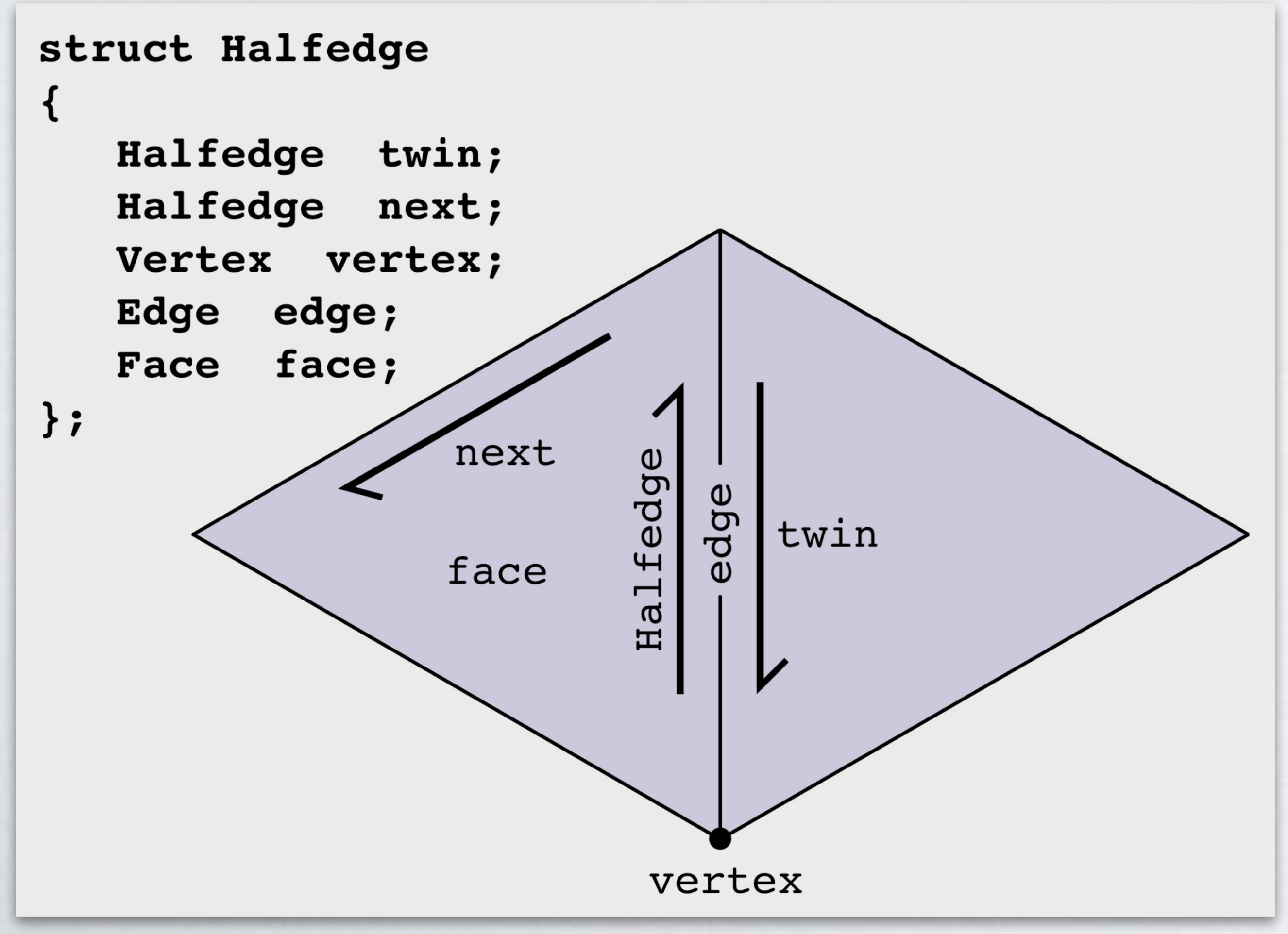



```
struct Vertex
{
   Halfedge  halfedge;
};
```





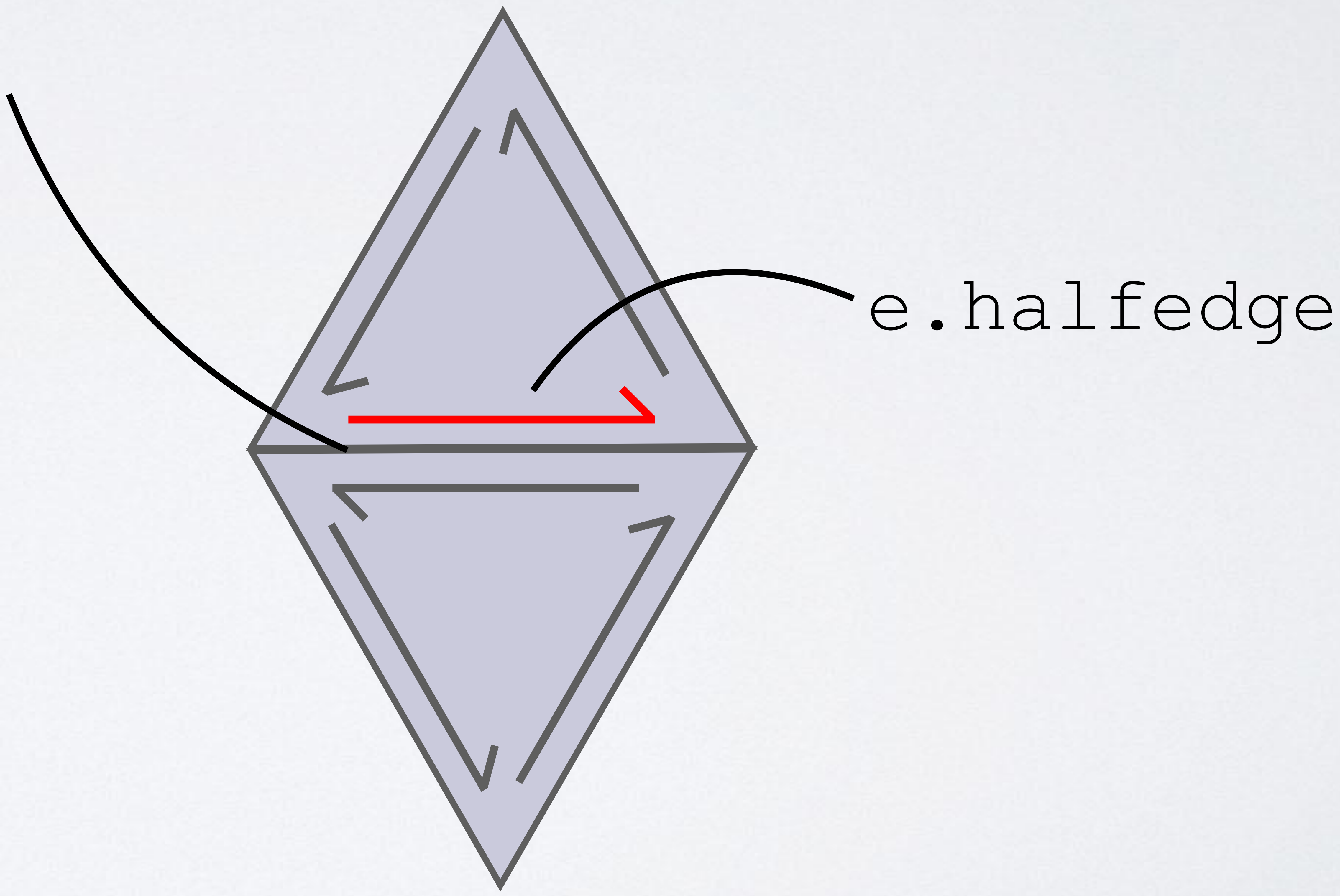


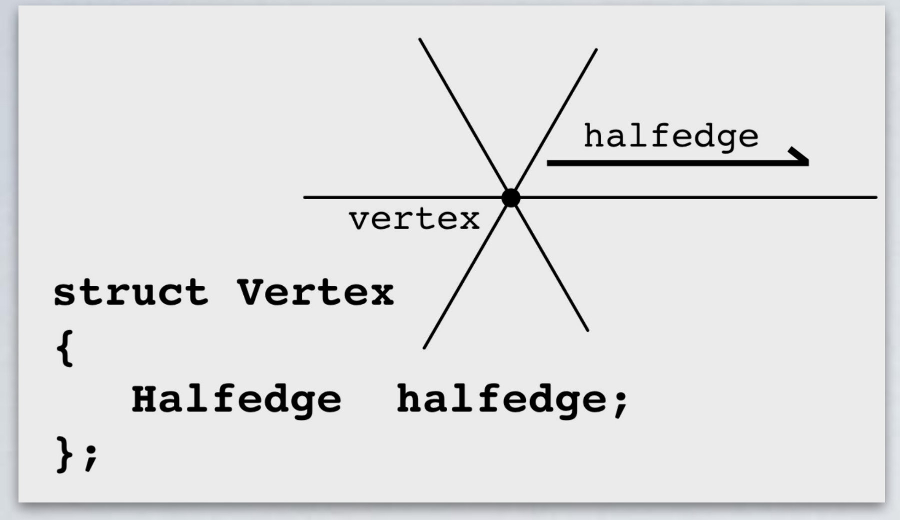


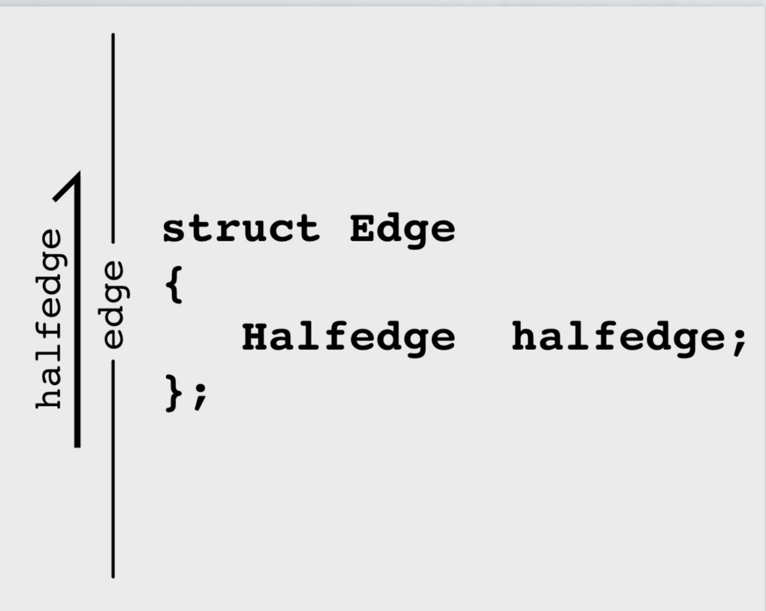
The Halfedge Data Structure

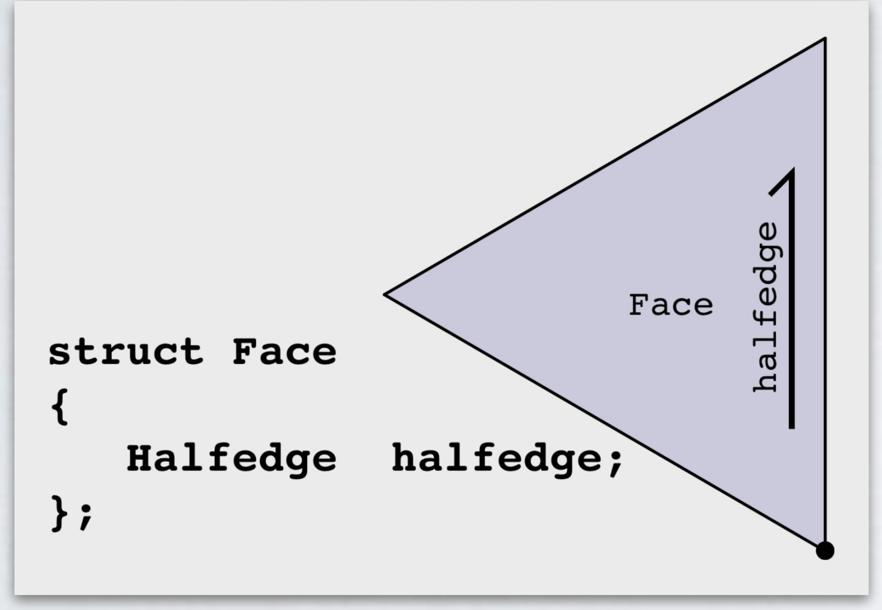
How would I find the faces adjacent to an edge?

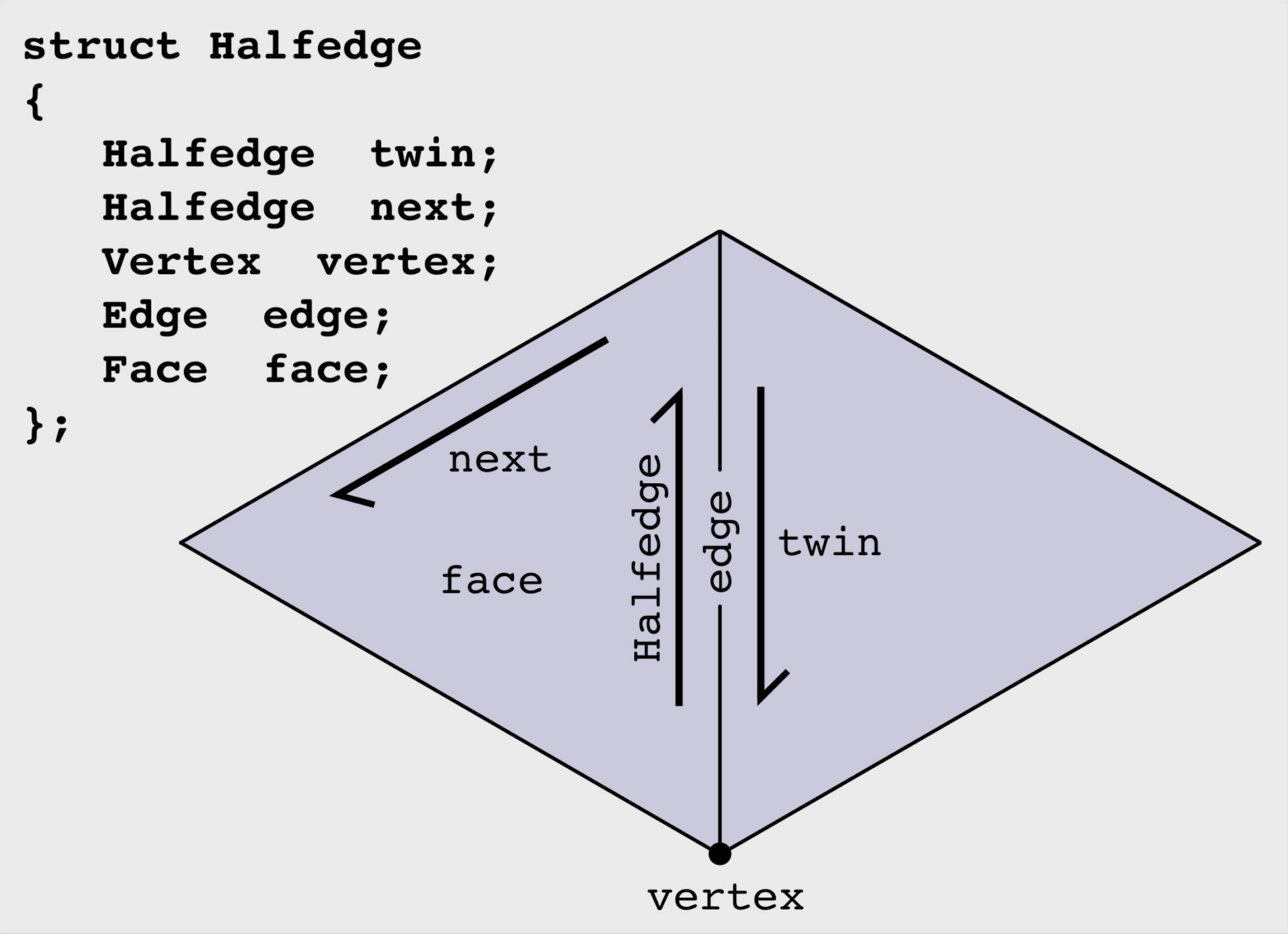
Given: `Edge e`









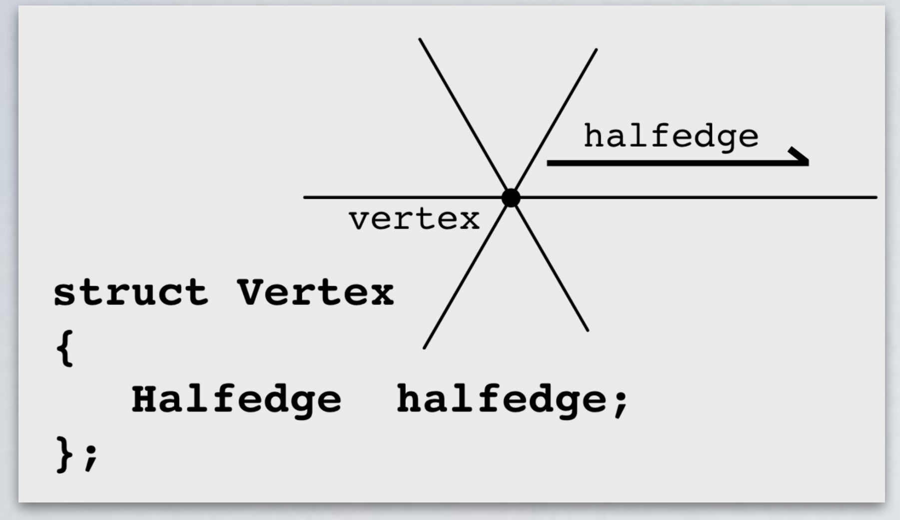


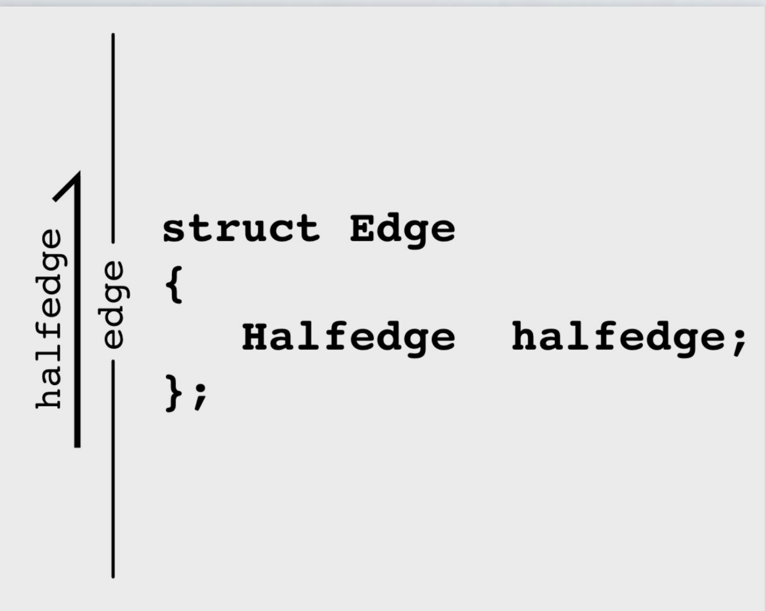
The Halfedge Data Structure

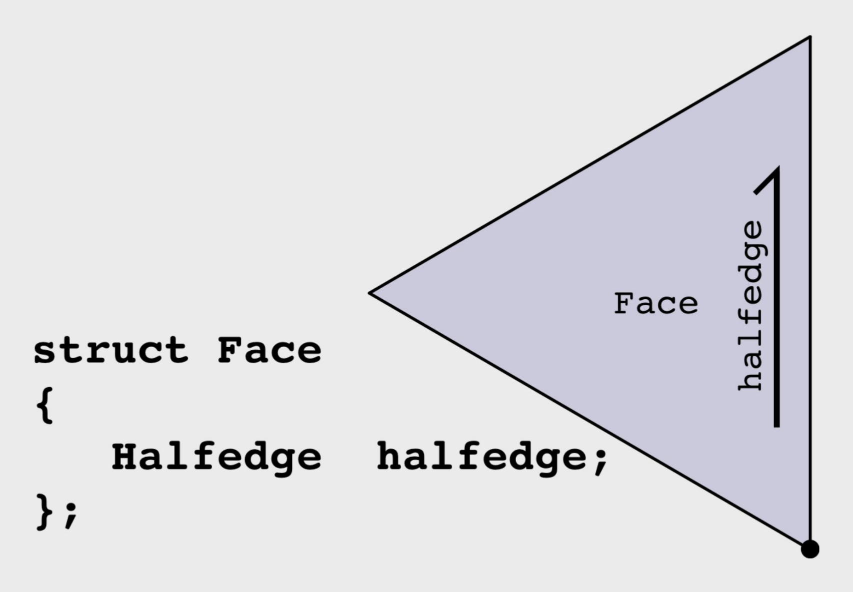
How would I find the faces adjacent to an edge?

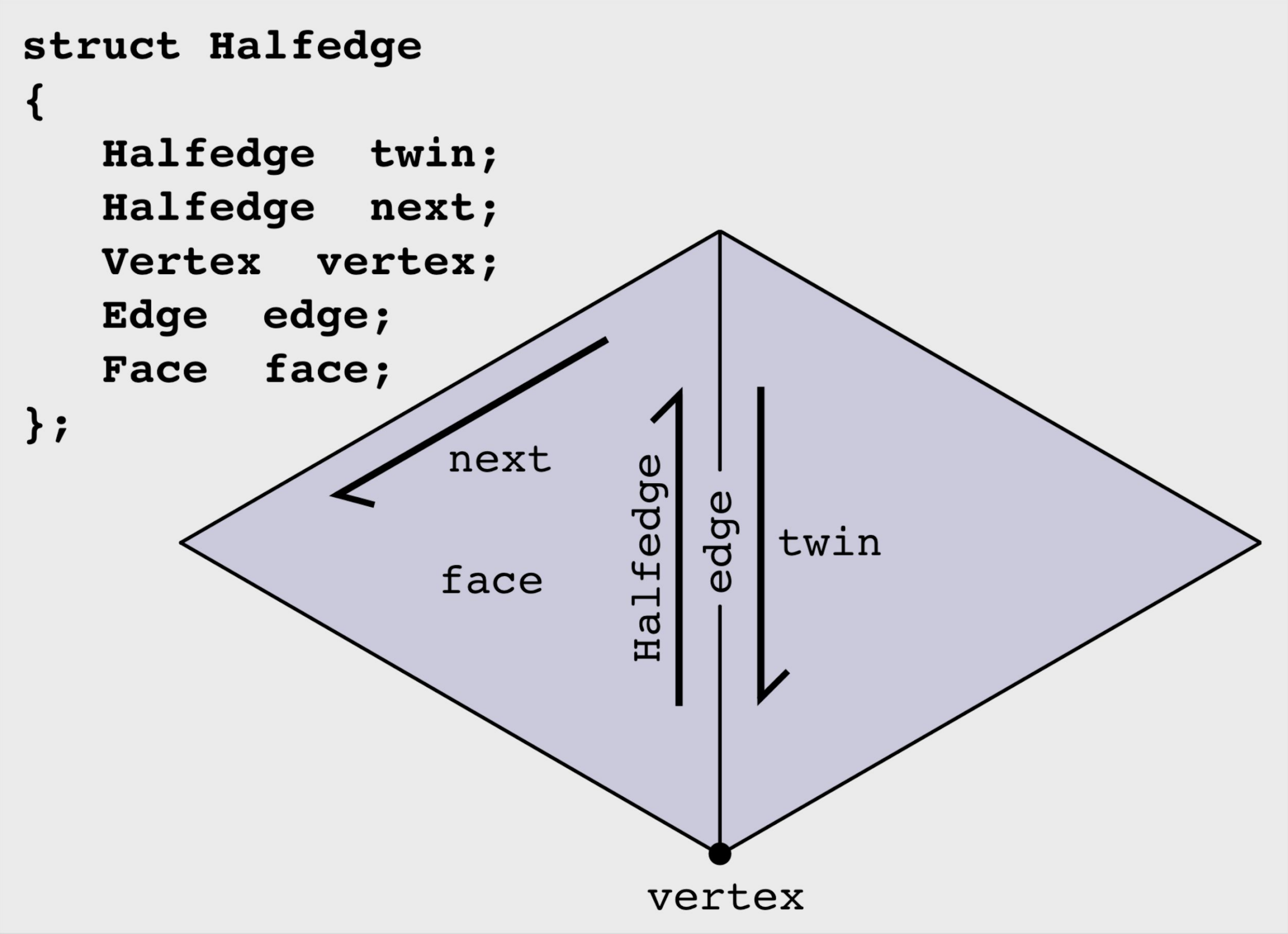
Given: `Edge e`

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





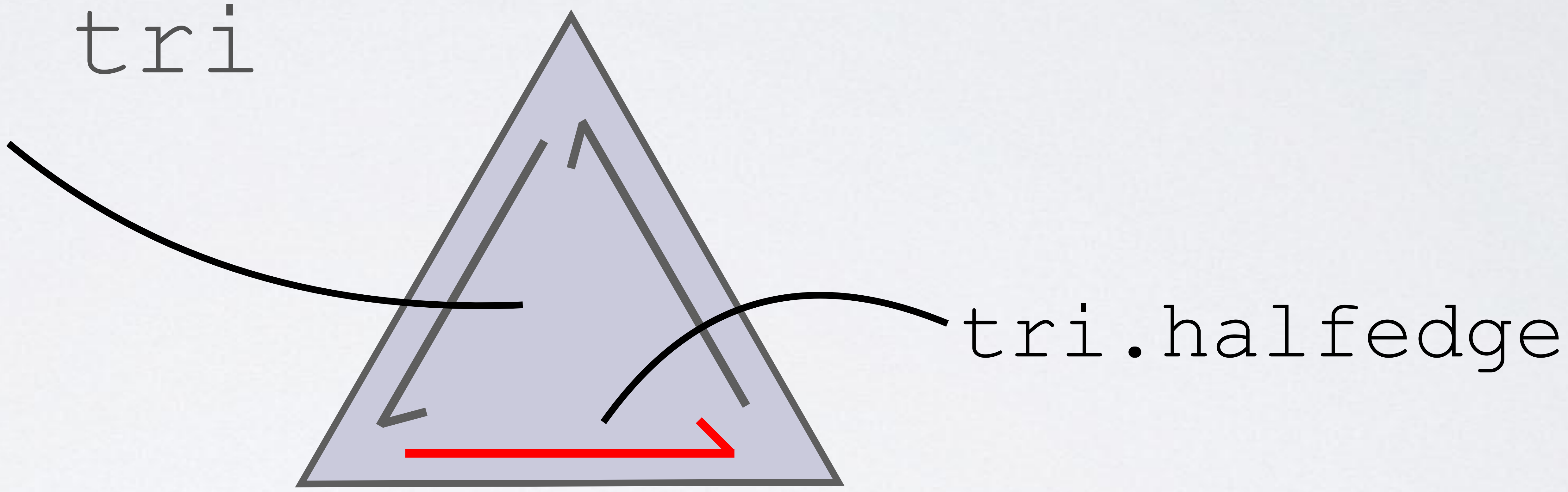


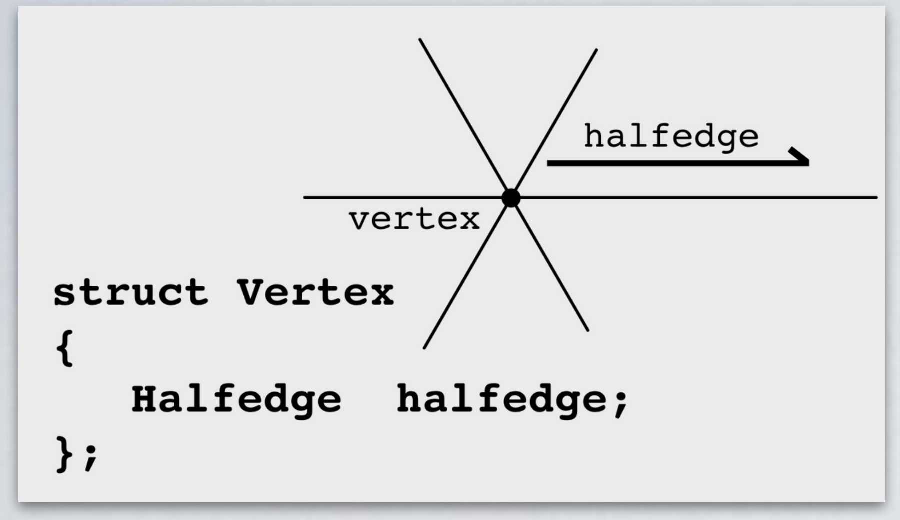


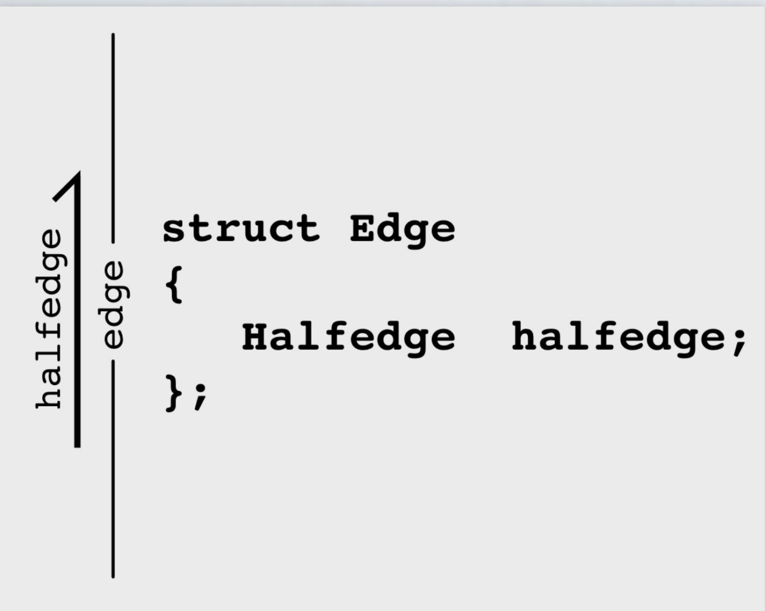
The Halfedge Data Structure

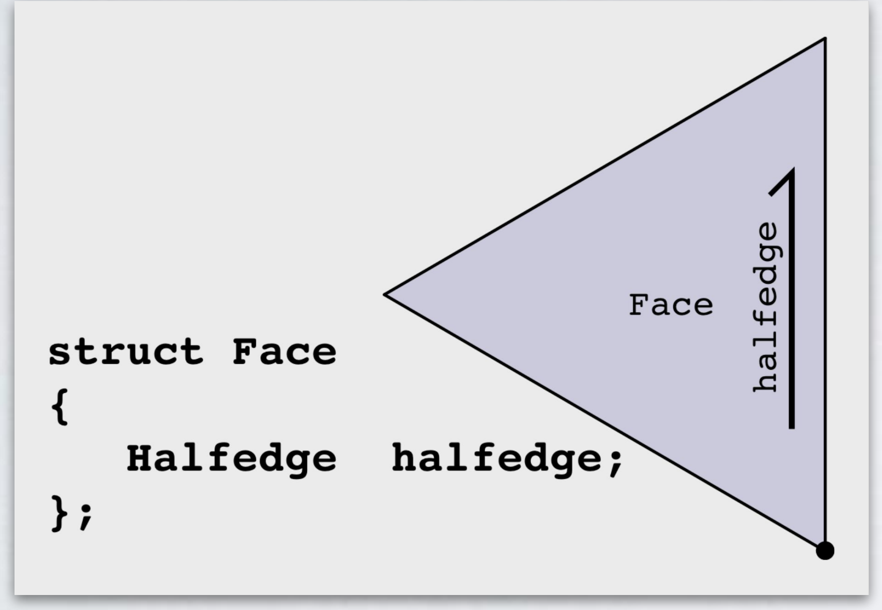
How would I find the edges adjacent to a triangle?

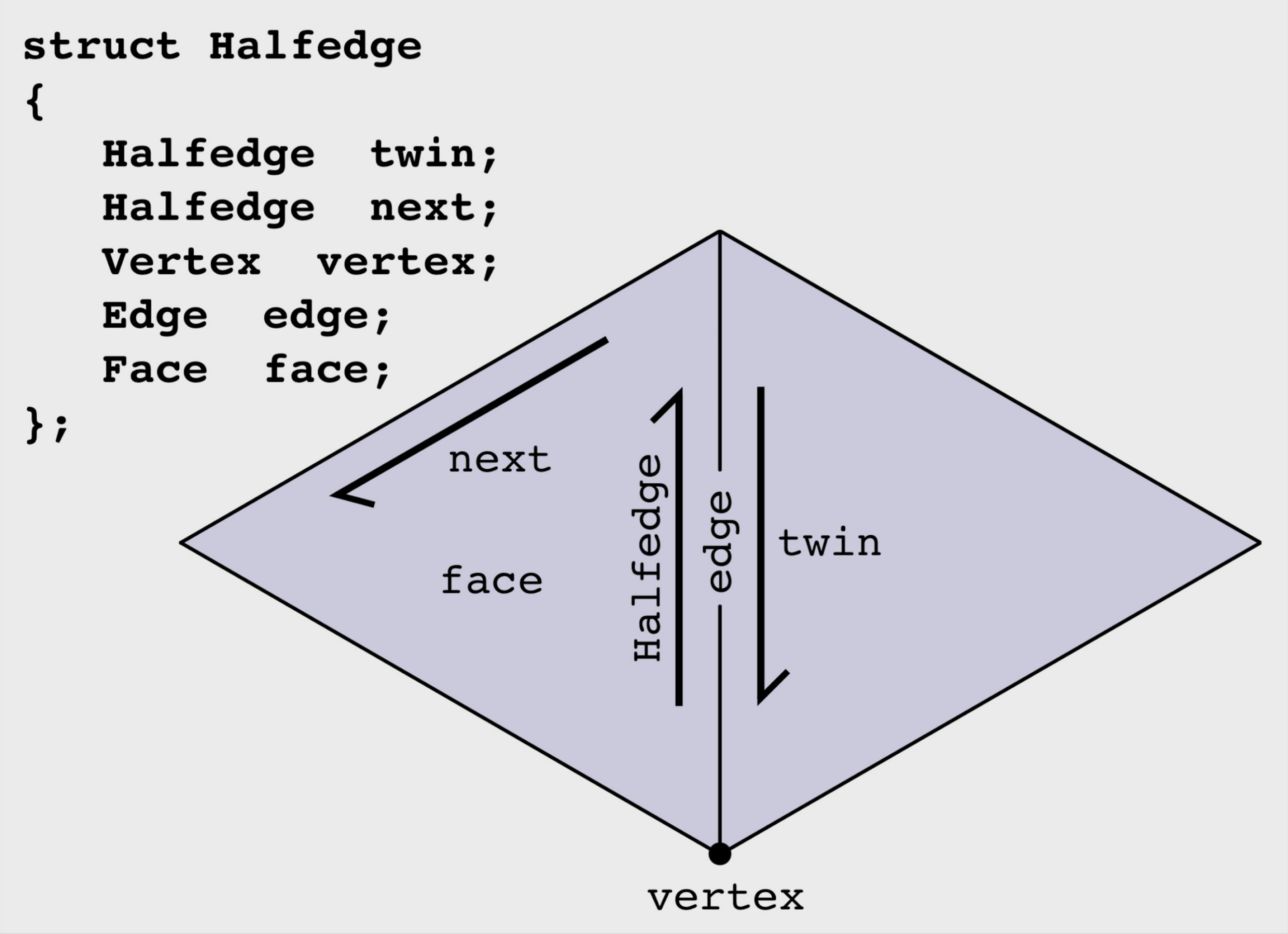
Given: `Face tri`









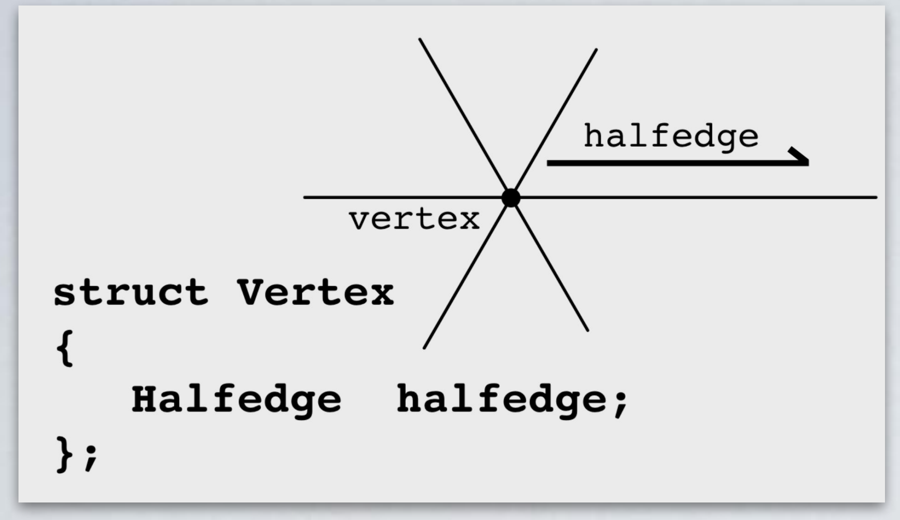


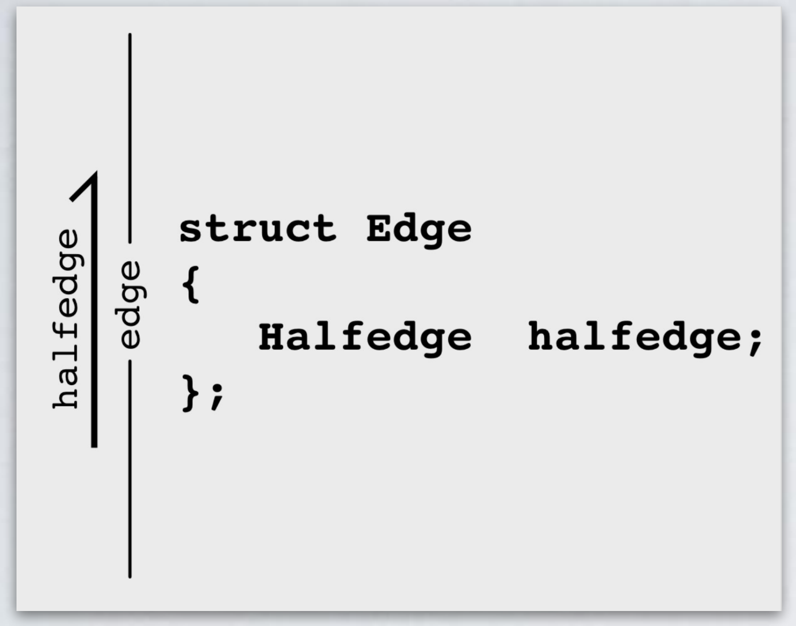
The Halfedge Data Structure

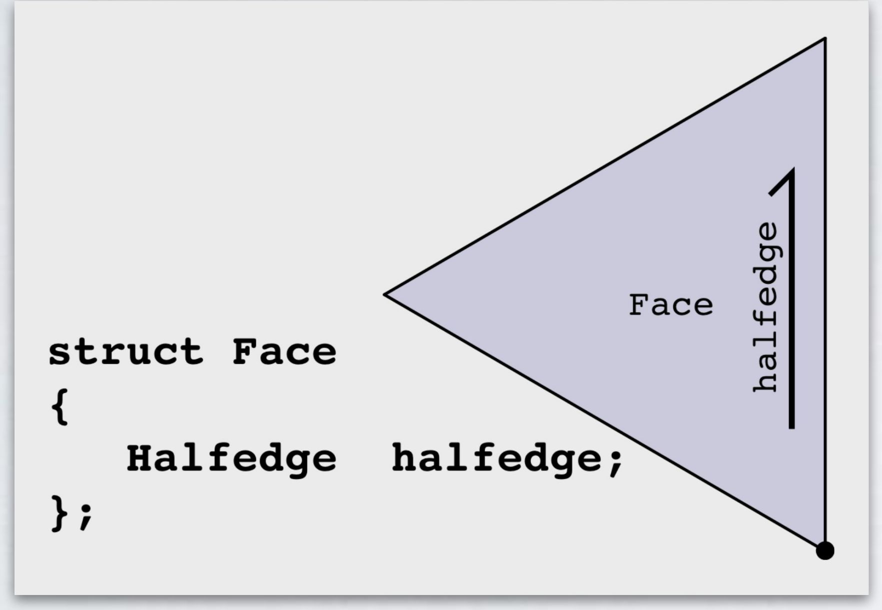
How would I find the edges adjacent to a triangle?

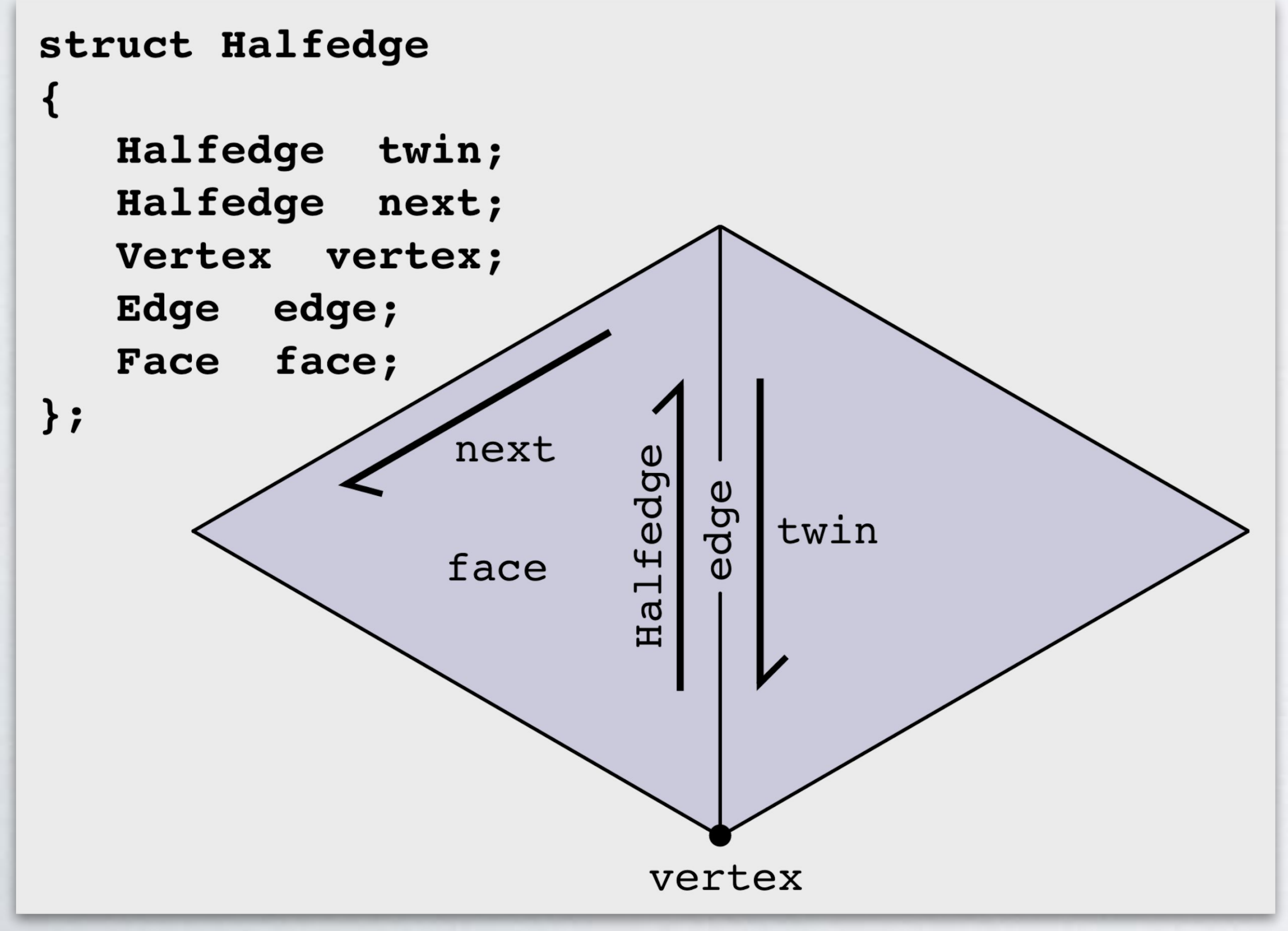
Given: `Face tri`

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





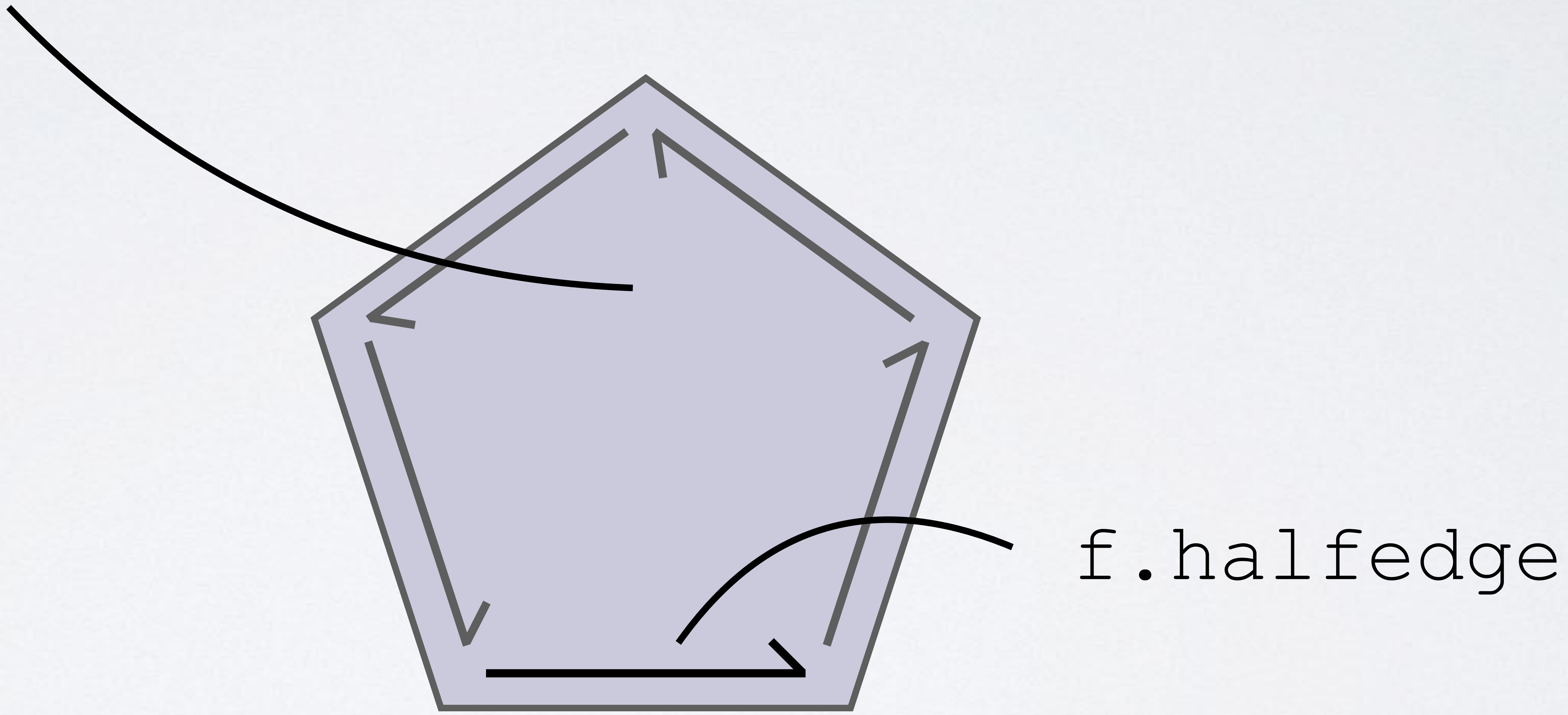




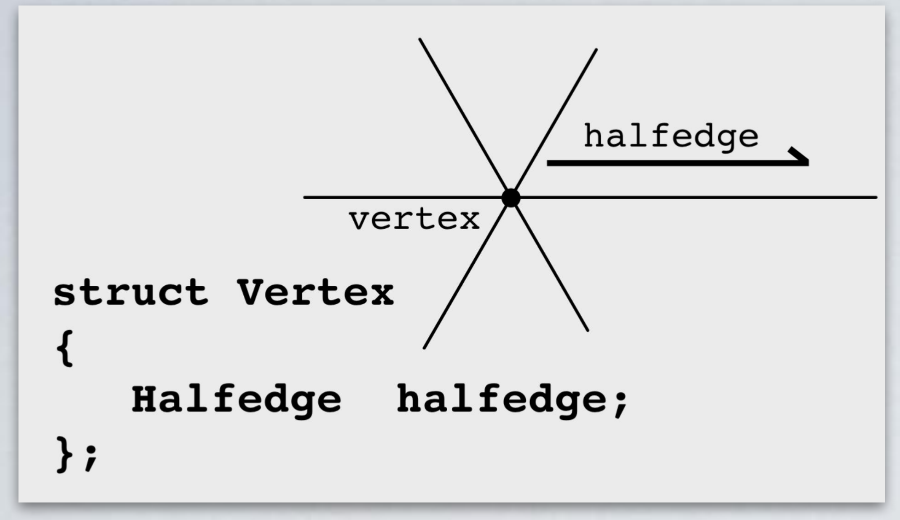
The Halfedge Data Structure

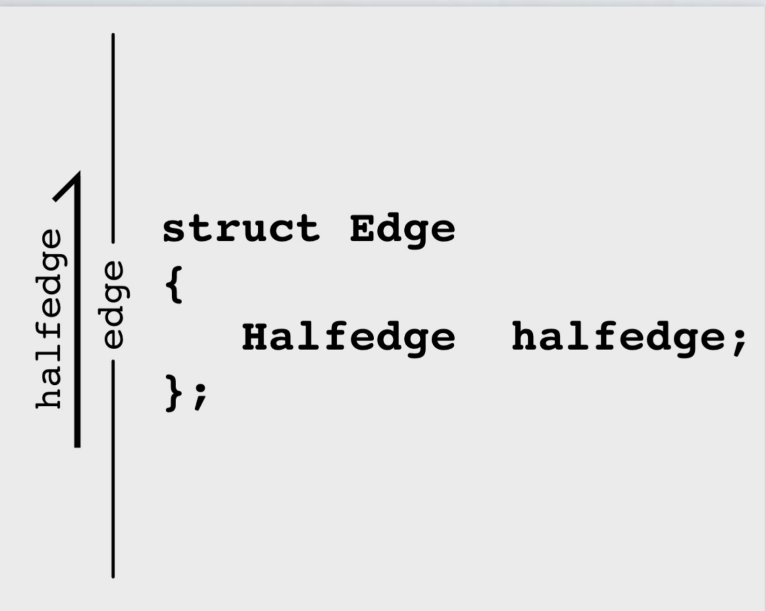
How would I loop over the edges adjacent to a polygon?

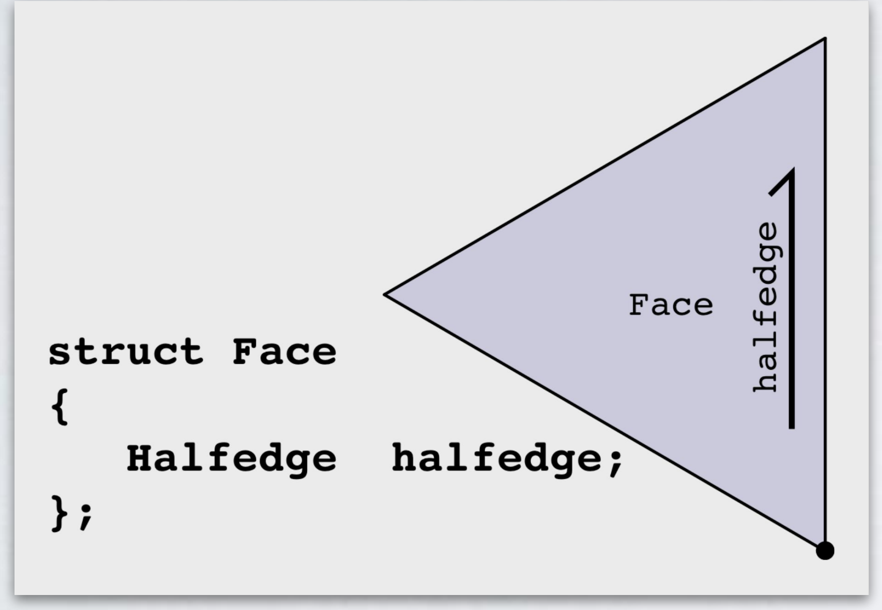
Given: `Face f`

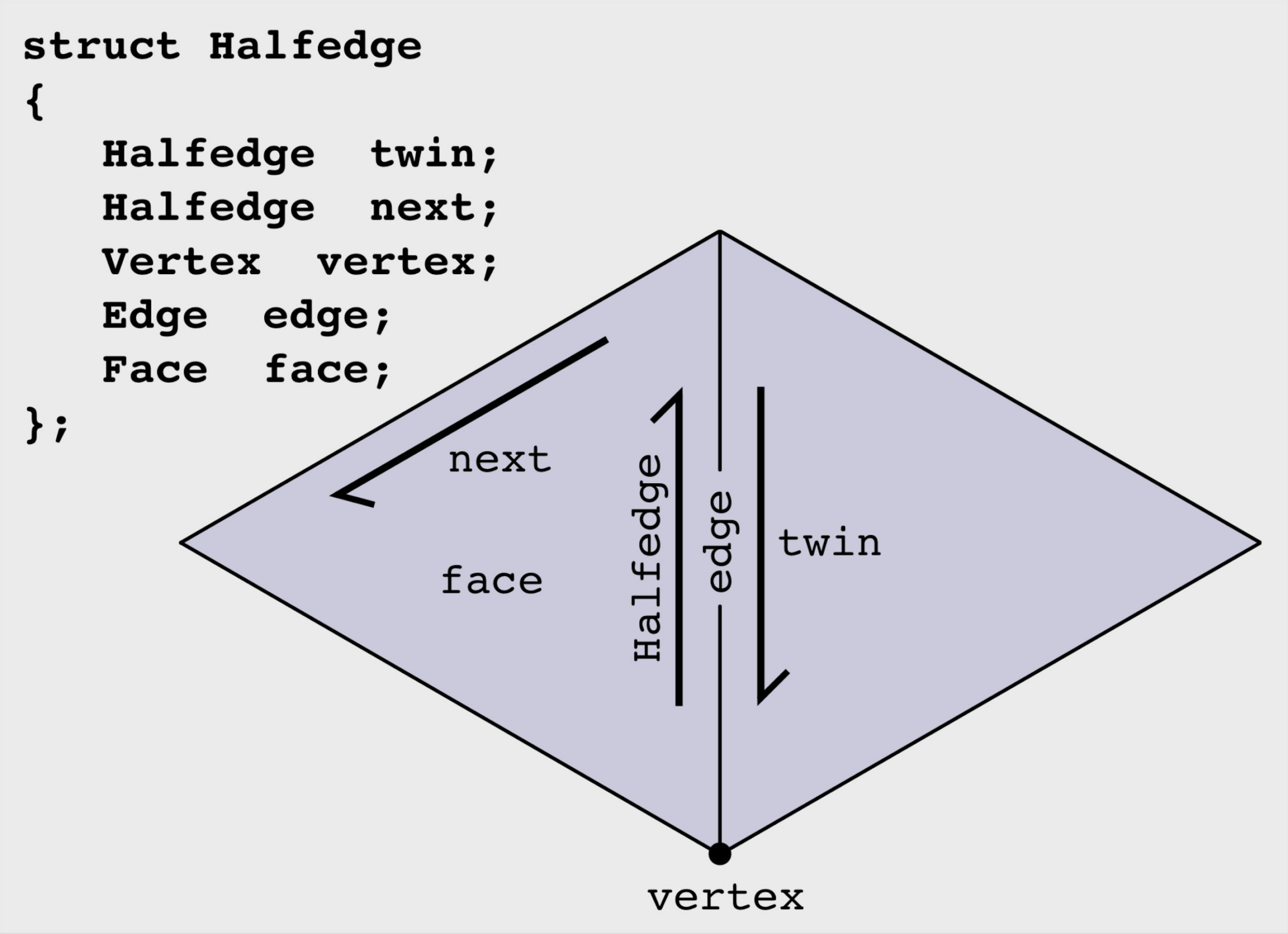


The Halfedge Data Structure





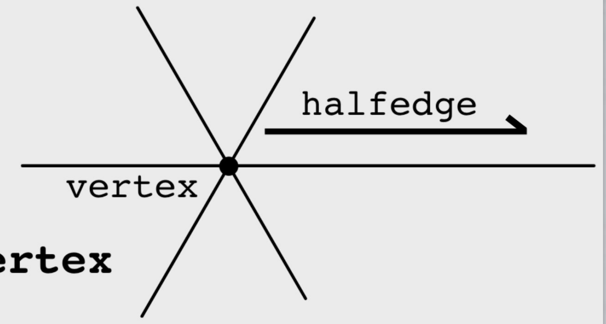




How would I loop over the edges adjacent to a polygon?

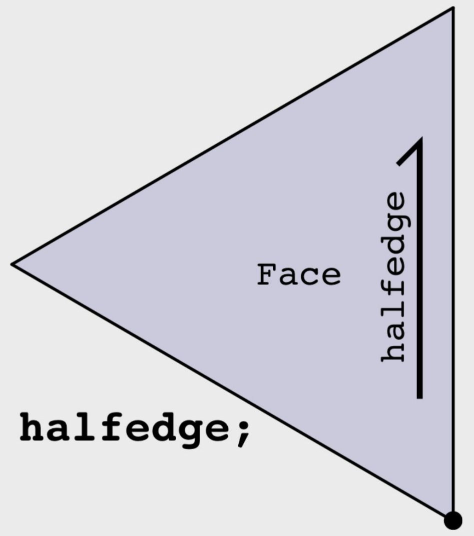
Given: `Face f`

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



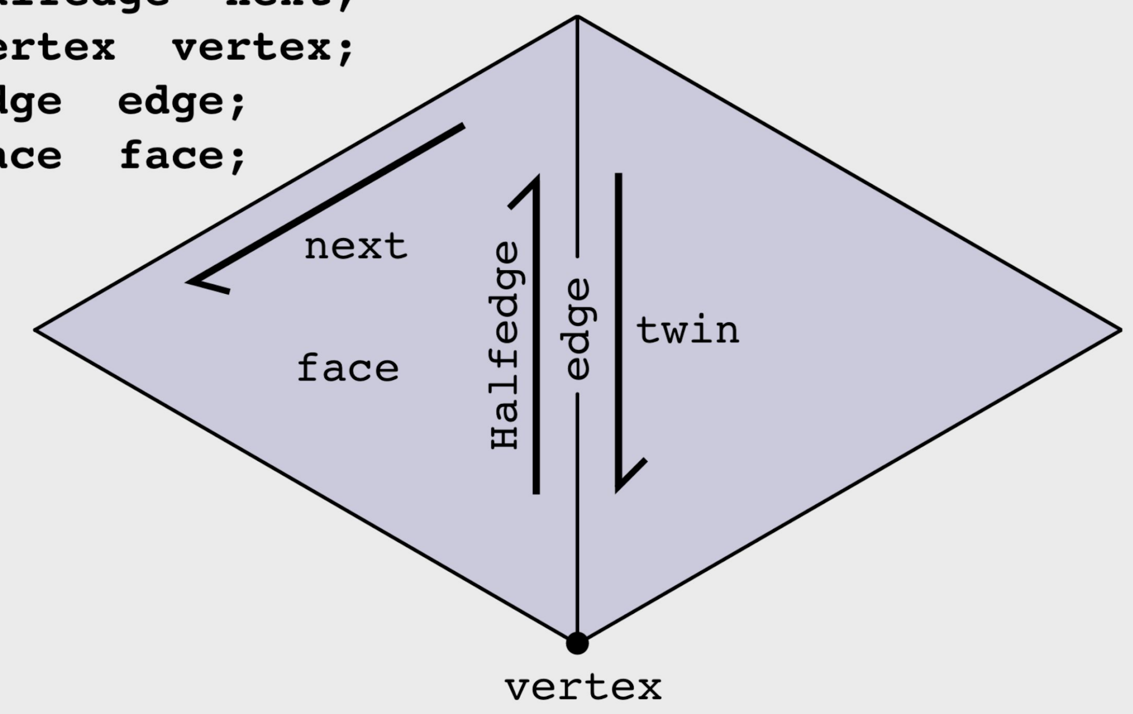

```
struct Vertex
{
   Halfedge  halfedge;
};
```


```
struct Edge
{
   Halfedge  halfedge;
};
```




```
struct Face
{
   Halfedge  halfedge;
};
```

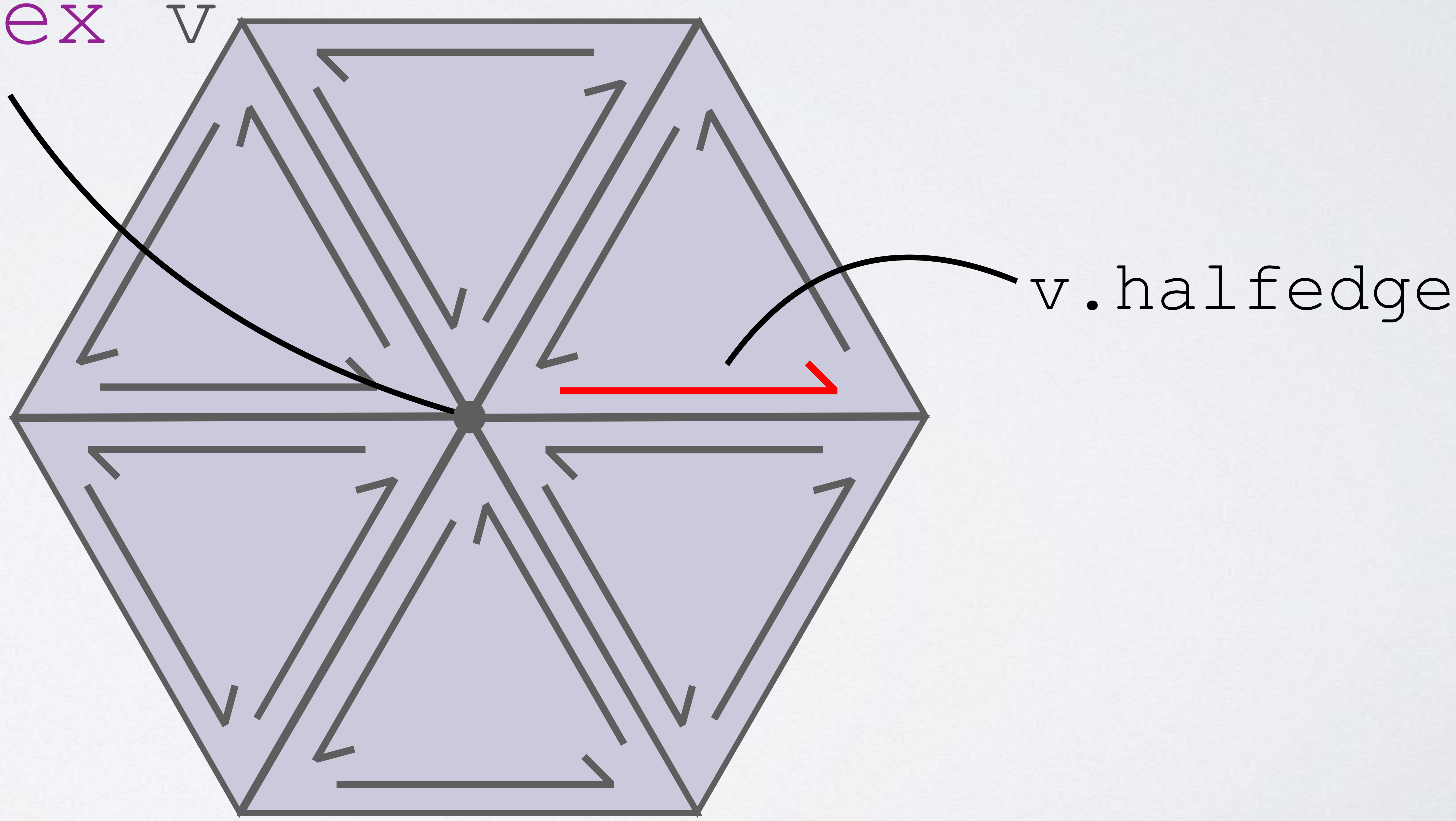
```
struct Halfedge
{
   Halfedge  twin;
   Halfedge  next;
   Vertex  vertex;
   Edge  edge;
   Face  face;
};
```

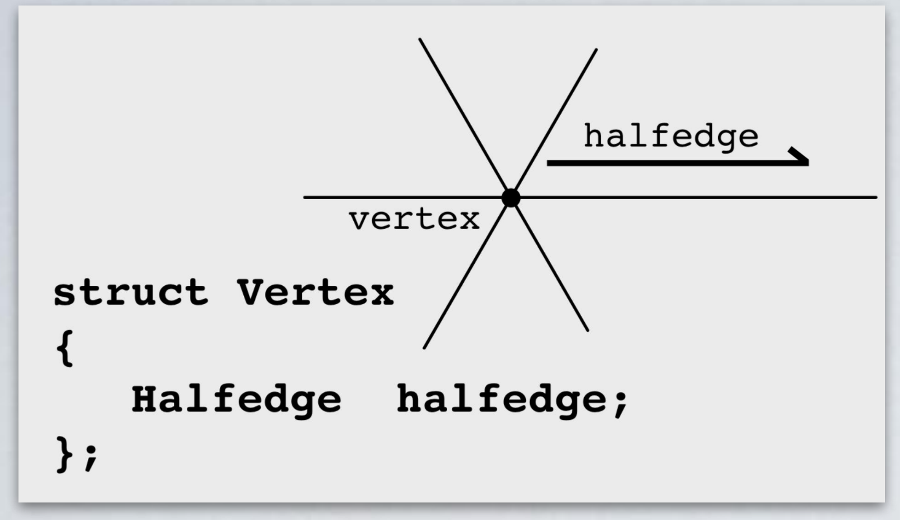


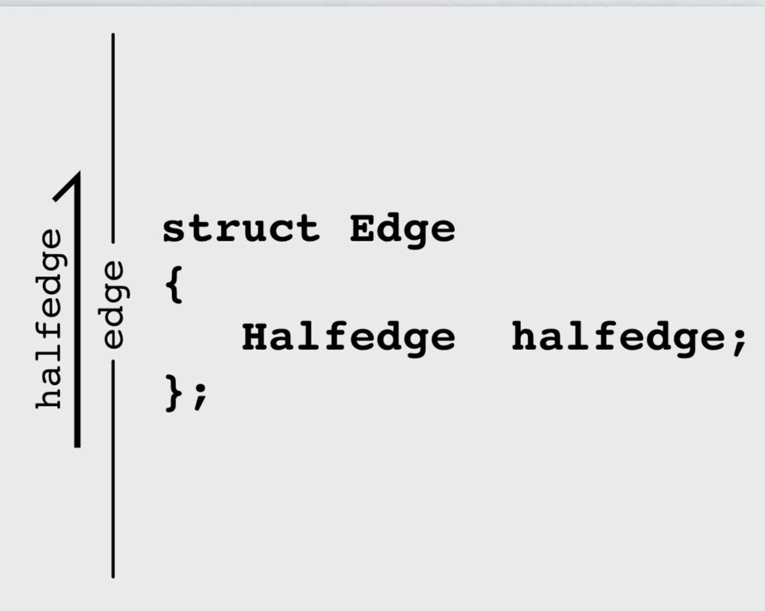
The Halfedge Data Structure

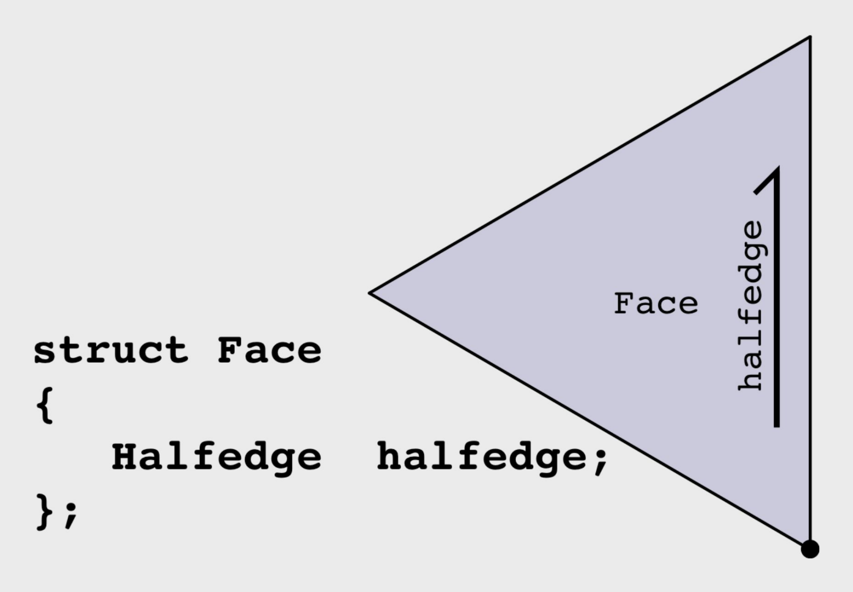
How would I loop over the edges adjacent to a vertex?

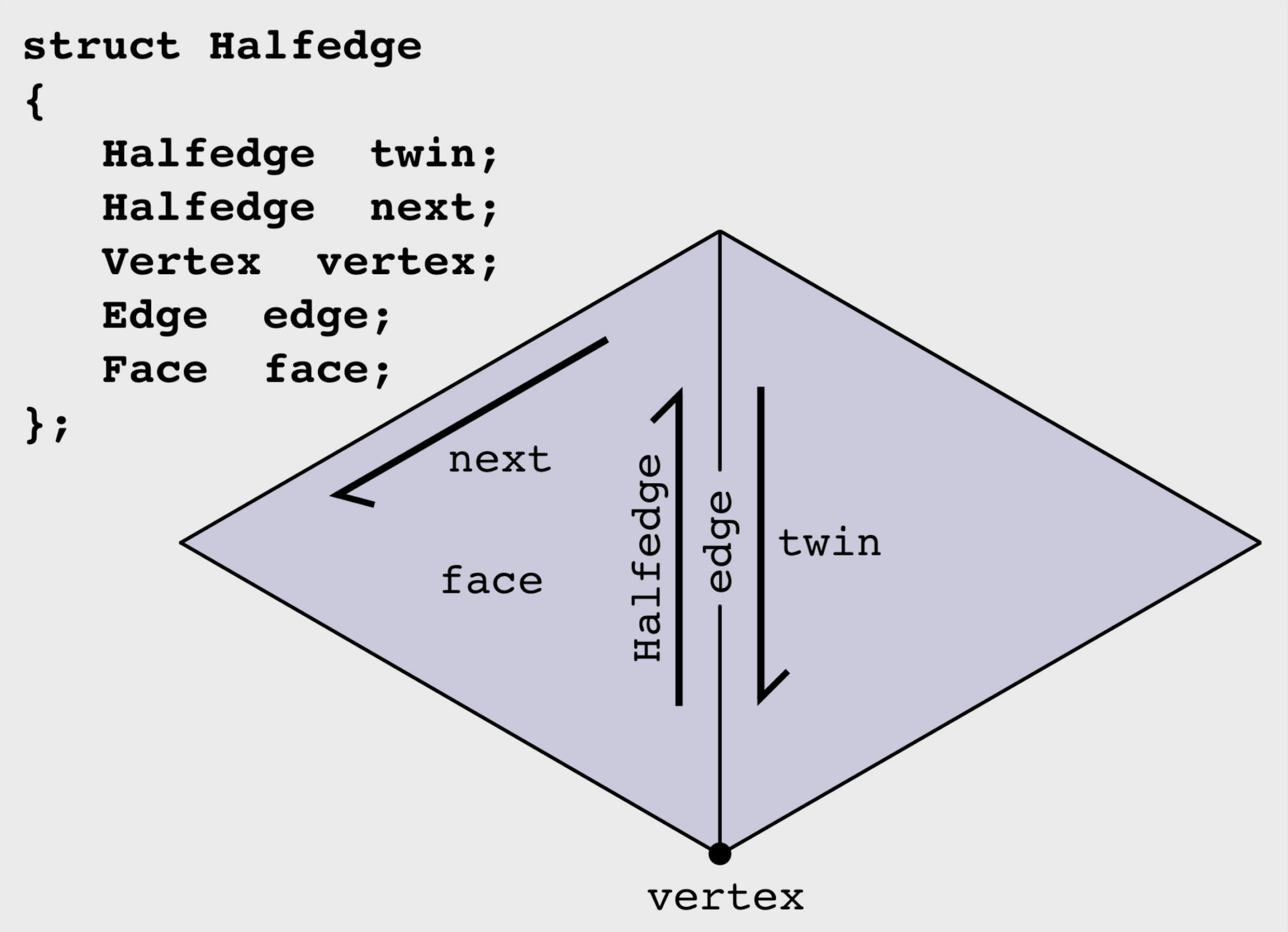
Given: `Vertex v`











The Halfedge Data Structure

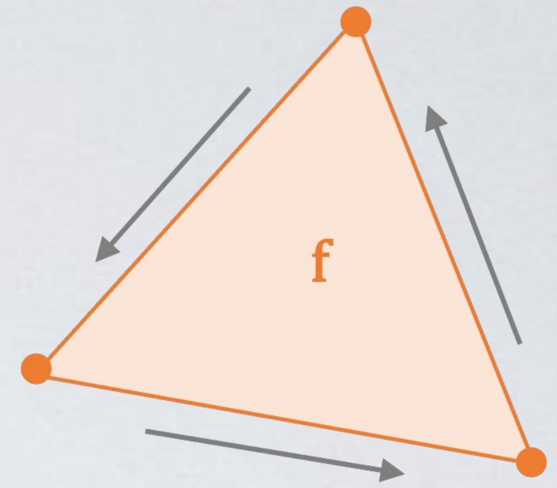
How would I loop over the edges adjacent to a vertex?

Given: `Vertex v`

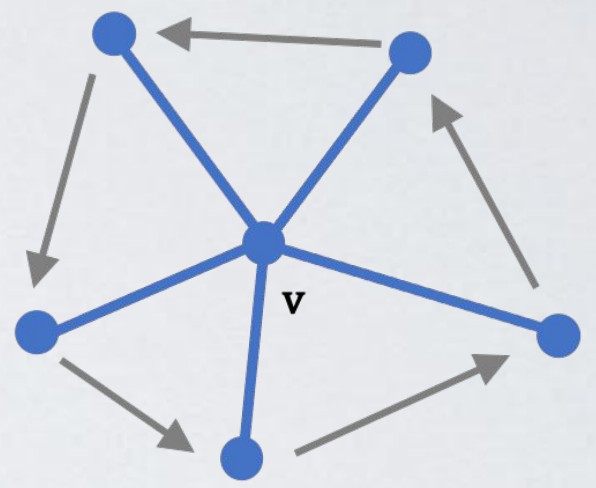
```
Halfedge start = v.halfedge;
Halfedge he = start;
do {
    Edge e = he.edge;
    /* Some code */
    he = he.twin.next;
} while (he != start);
```

Many convenience functions in both JS and C++!

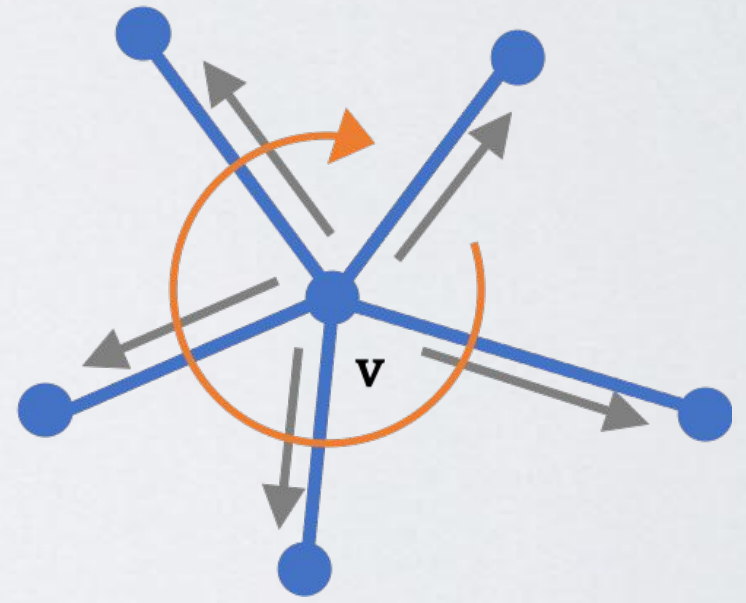
`f.adjacentVertices()` → iterator over vertices adjacent to face f



`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

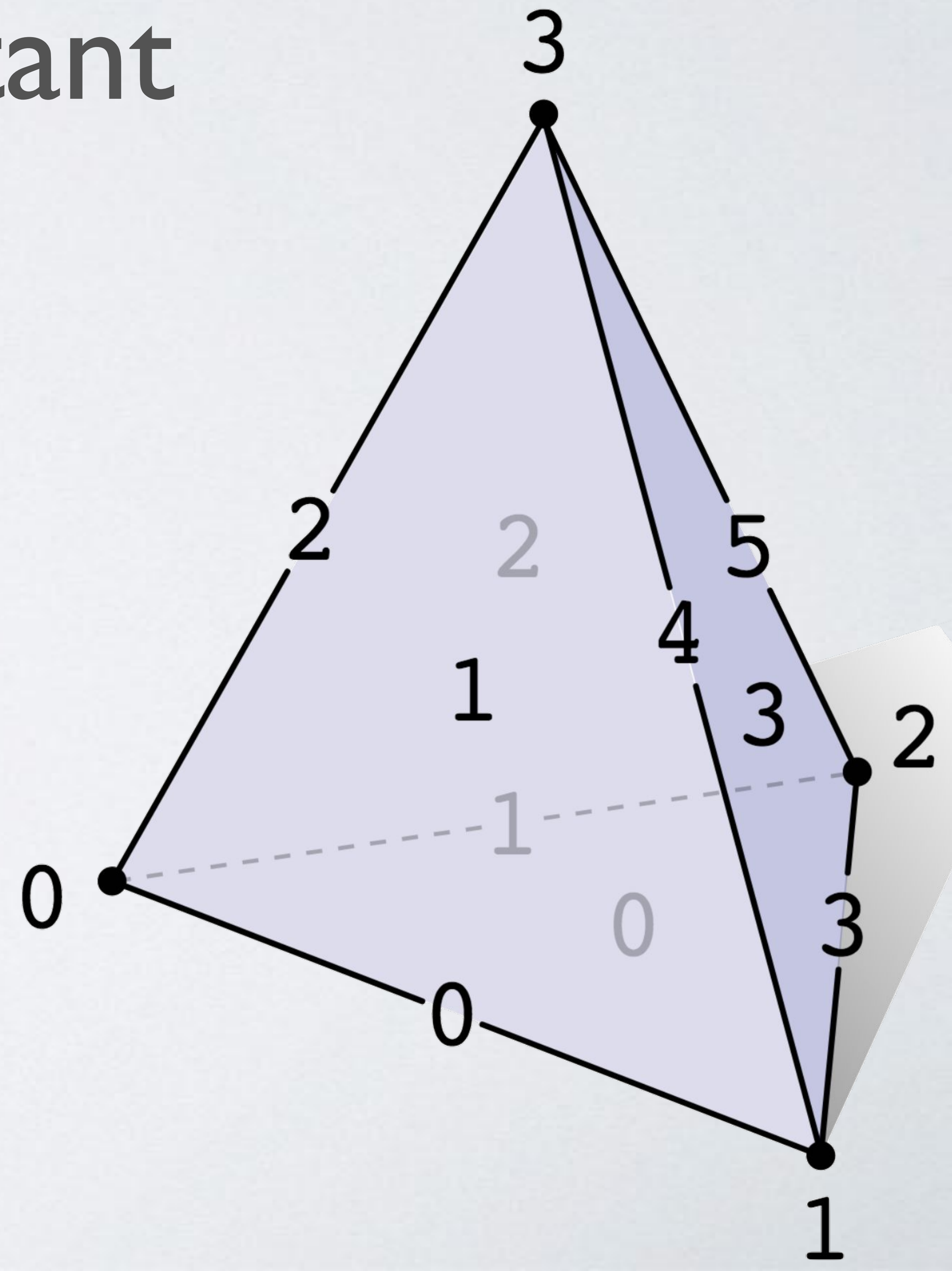
Matrices

- What matrices do we care about?

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

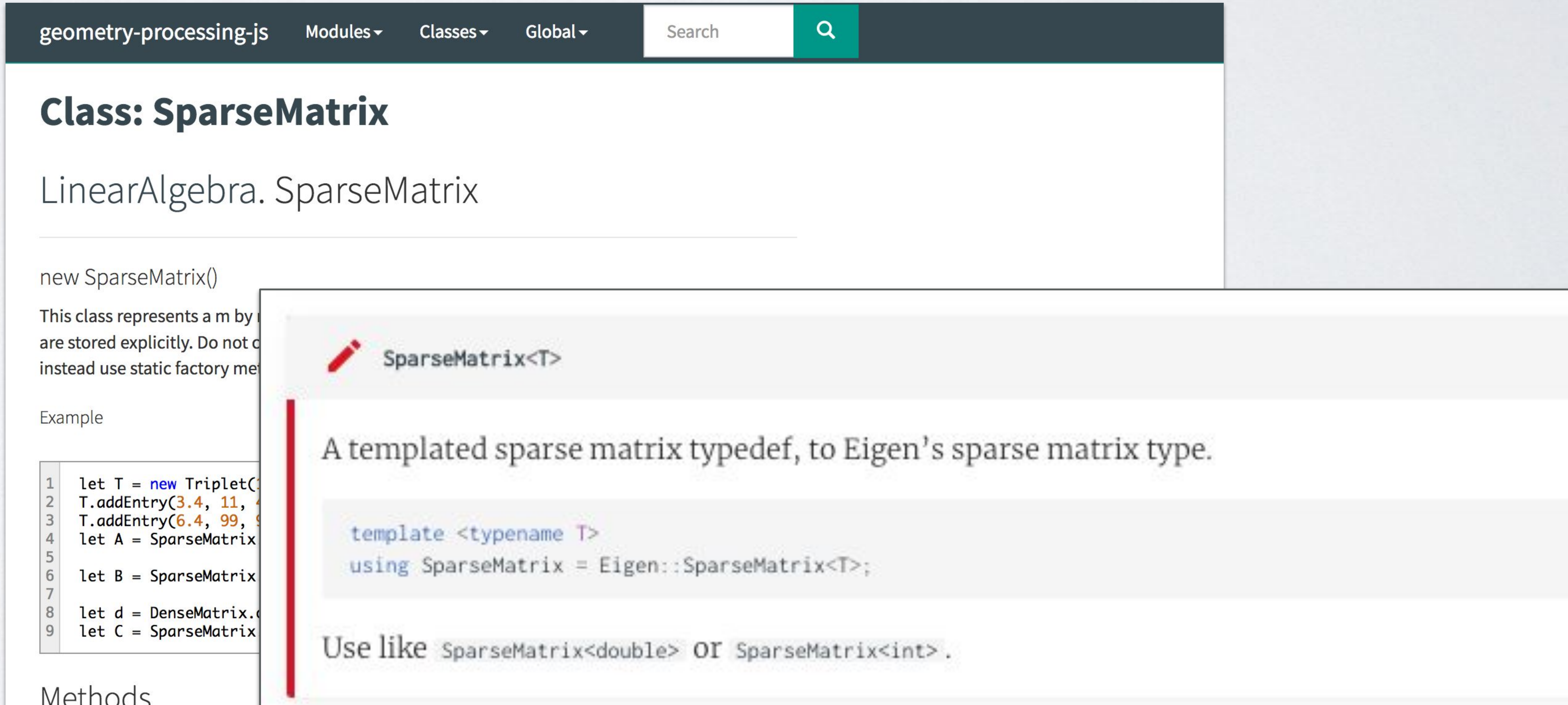
$$E^0 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

$$E^1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \end{array}$$



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

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

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`

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

```
 A = [ 5 8 3 6 ]
IA = [ 0 0 2 3 4 ]
JA = [ 0 1 2 1 ]
```

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

$$\begin{aligned} x + 2y - 4z &= 1 \\ 3x - 5y + 7z &= 2 \\ -x + 3y + 5z &= -2 \end{aligned} \;\rightarrow\; \underbrace{\begin{pmatrix} 1 & 2 & -4 \\ 3 & -5 & 7 \\ -1 & 3 & 5 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x \\ y \\ z \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 1 \\ 2 \\ -2 \end{pmatrix}}_{b} \;\rightarrow\; Ax = b$$

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?

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

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

- (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 *LLT* 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

$$Q^{-1} = Q^T$$

$$Qx = b \implies x = Q^T b$$

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

$$Ax = b$$

$$QRx = b$$

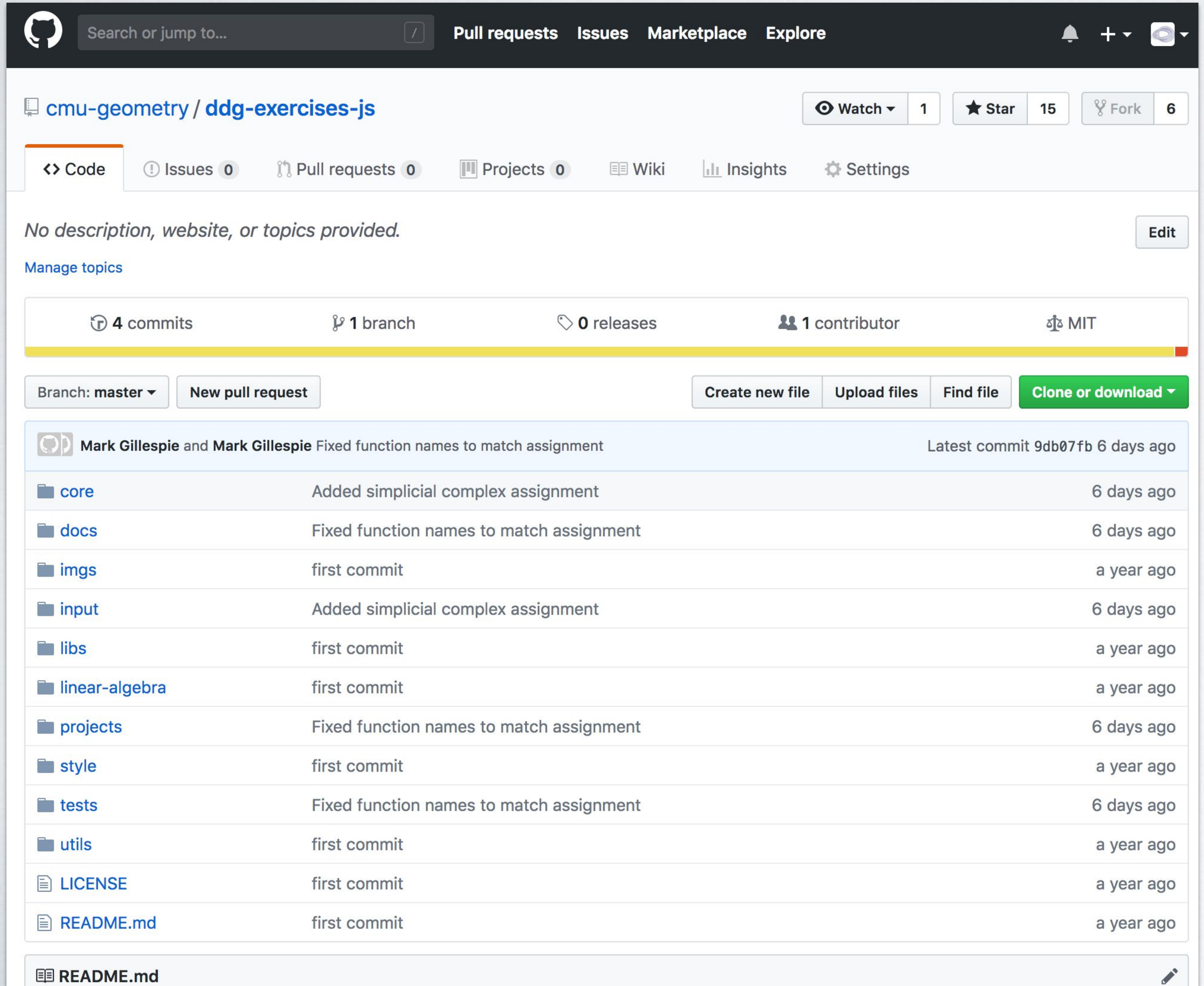
$$Qy = b \text{ and } y = Rx$$

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
- 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`

geometry-processing-js Modules▾ Classes▾ Global▾ Searc

ddg-exercises-js

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

– raw

```
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

Documentation

geometry-processing-js Modules▾ Classes▾ Global▾ Search

Class: Mesh

Core. Mesh

new Mesh()

This class represents a Mesh.

Properties:

Name	Type	Descripti
vertices	Array.<module:Core.Vertex>	The vertic
edges	Array.<module:Core.Edge>	The edges
faces	Array.<module:Core.Face>	The faces
corners	Array.<module:Core.Corner>	The corne
halfedges	Array.<module:Core.Halfedge>	The halfe
boundaries	Array.<module:Core.Face>	The boun mesh.
generators	Array.<Array.<module:Core.Halfedge>>	An array o [[h11, h21 represent homology

Methods

geometry-processing-js Modules▾ Classes▾ Global▾ Search

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

```
let T = new Triplet(100, 100);
T.addEntry(3.4, 11, 43);
T.addEntry(6.4, 99, 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.

Parameters:

Name Type Description

geometry-processing-js Modules▾ Classes▾ Global▾ Sear

ddg-exercises-js

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

– raw

```
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).

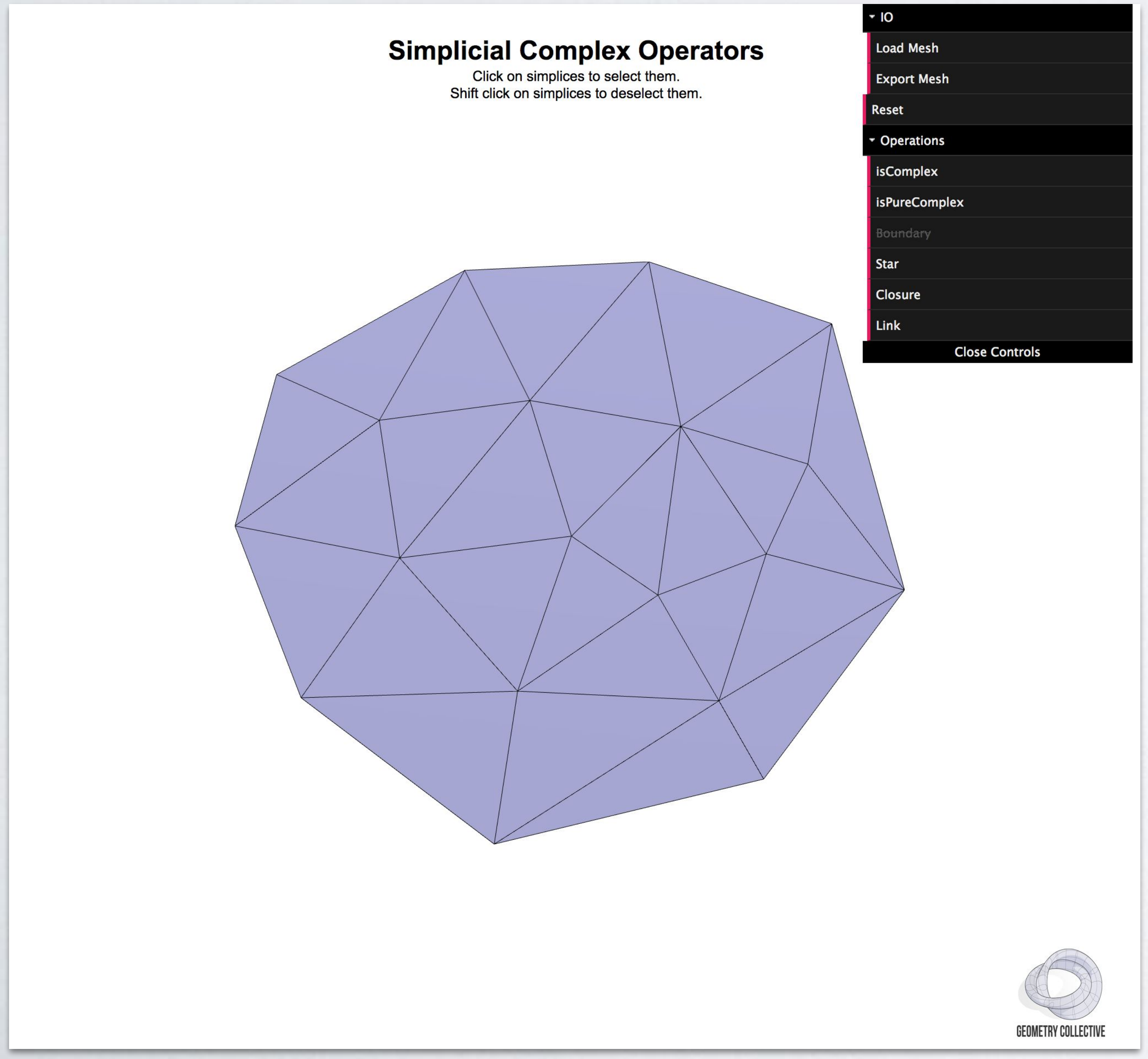
Dependencies (all included)

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

Coding Assignments

- Viewers

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

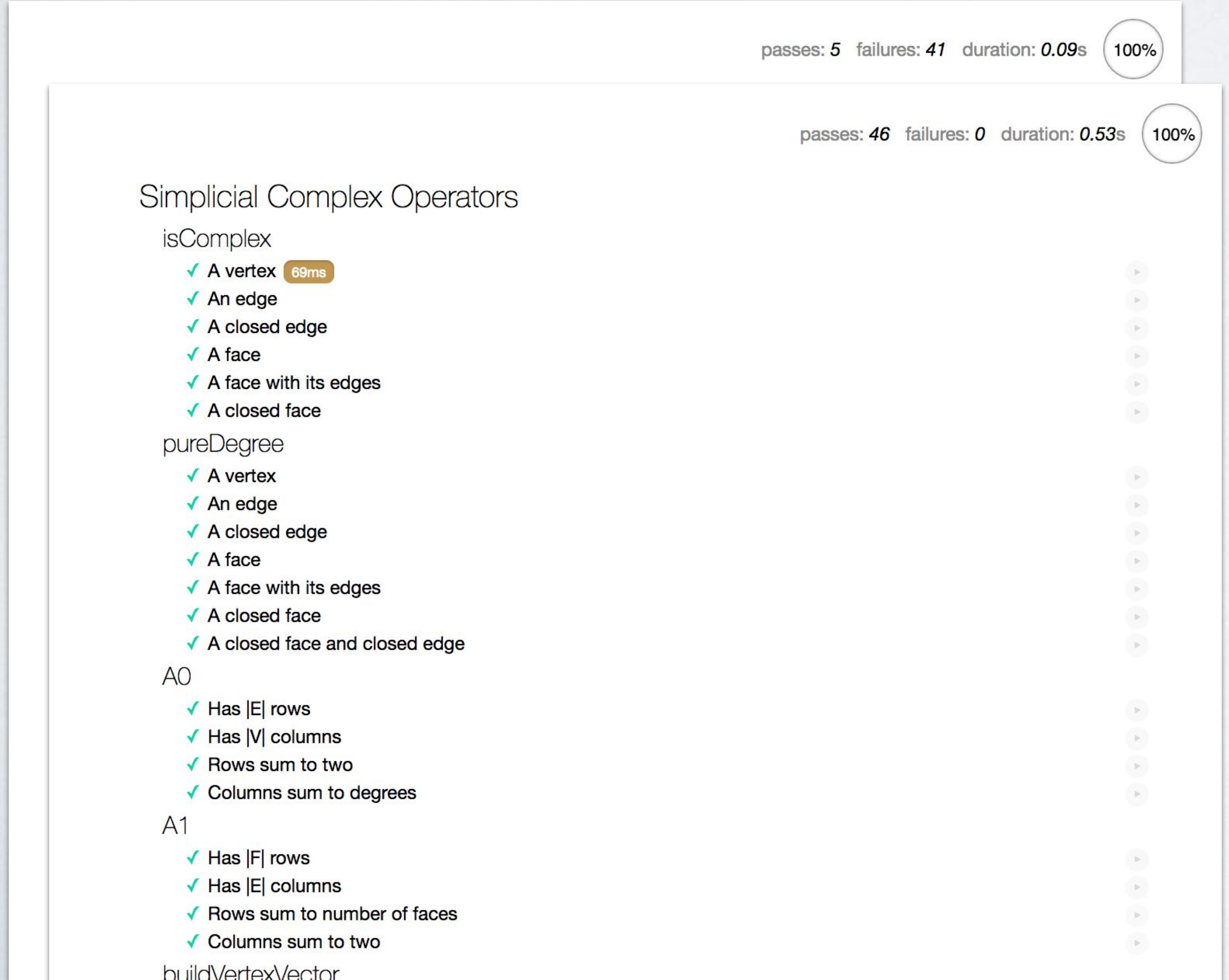


- Write code in project folder or one of the modules
- Graphics programming often involves a lot of boilerplate before getting started drawing - We've mostly done that for you. You just have to fill in the interesting bits

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

geometry-processing-js Modules▾ Classes▾ Global▾ Search

Class: Mesh

Core. Mesh

new Mesh()

This class represents a Mesh.

Properties:

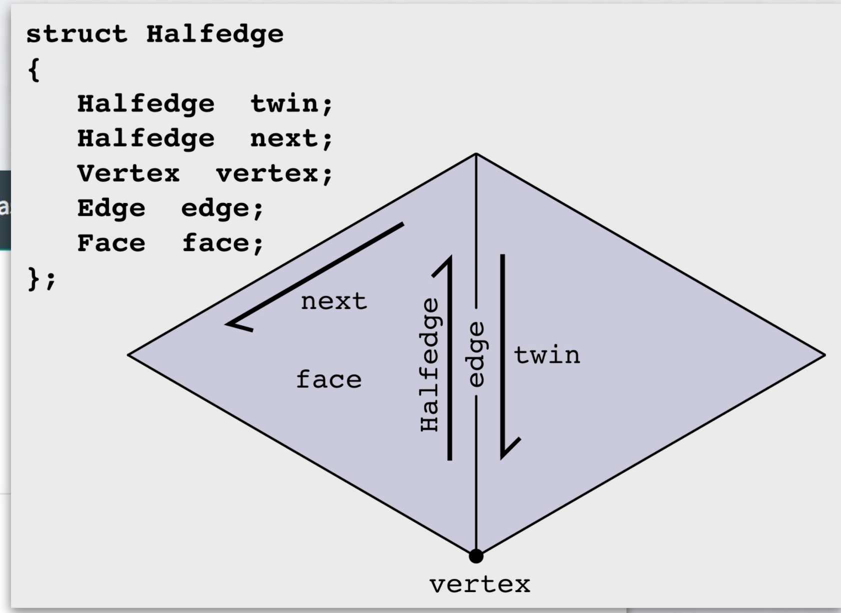
Name	Type	Description
vertices	Array.<module:Core.Vertex>	The vertices contained in this mesh.
edges	Array.<module:Core.Edge>	The edges contained in this mesh.
faces	Array.<module:Core.Face>	The faces contained in this mesh.
corners	Array.<module:Core.Corner>	The corners contained in this mesh.
halfedges	Array.<module:Core.Halfedge>	The halfedges contained in this mesh.
boundaries	Array.<module:Core.Face>	The boundary loops contained in this mesh.
generators	Array.<Array.<module:Core.Halfedge>>	An array of halfedge arrays, i.e., [[h11, h21, ..., hn1], [h12, h22, ..., hm2], ...] representing this mesh's homology generators.

Methods

geometry-processing-js Modules▾ Cla

Class: Halfedge

Core. Halfedge



new Halfedge()

This class defines the connectivity of a Mesh.

Properties:

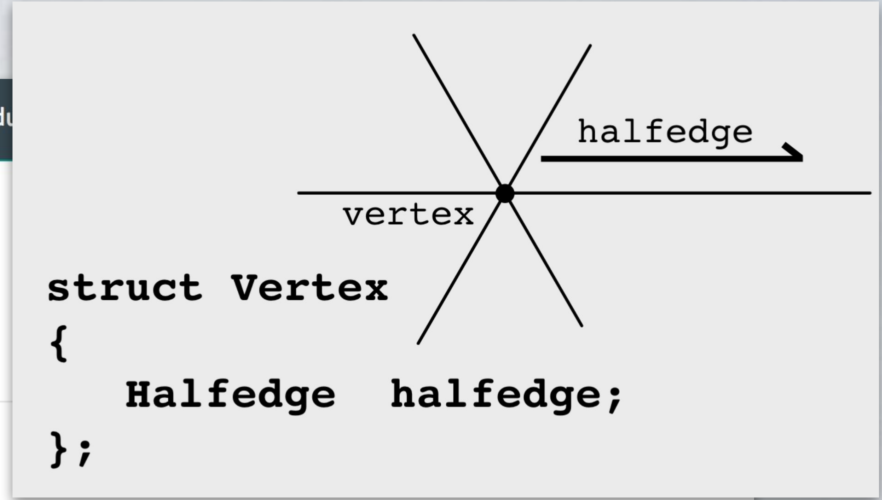
Name	Type	Description
vertex	module:Core.Vertex	The vertex at the base of this halfedge.
edge	module:Core.Edge	The edge associated with this halfedge.
face	module:Core.Face	The face associated with this halfedge.
corner	module:Core.Corner	The corner opposite to this halfedge. Undefined if this halfedge is on the boundary.
next	module:Core.Halfedge	The next halfedge (in CCW order) in this halfedge's face.
prev	module:Core.Halfedge	The previous halfedge (in CCW order) in this halfedge's face.
twin	module:Core.Halfedge	The other halfedge associated with this halfedge's edge.
onBoundary	boolean	A flag that indicates whether this halfedge is on a boundary.

Documentation generated by JSDoc 3.5.5 on Tue Jan 22nd 2019 using the DocStrap te

geometry-processing-js Modu

Class: Vertex

Core. Vertex



new Vertex()

This class represents a vertex in a Mesh.

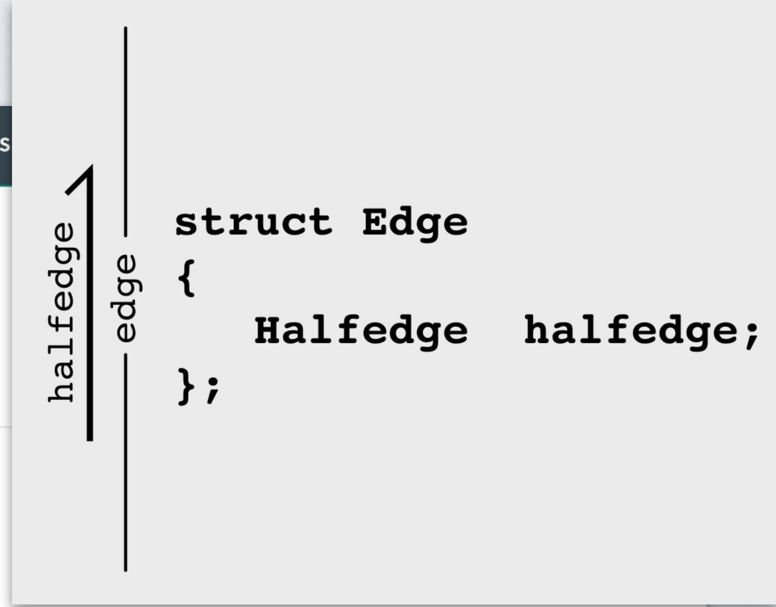
Properties:

Name	Type	Description
halfedge	module:Core.Halfedge	One of the outgoing halfedges associated with this vertex.

geometry-processing-js Modules

Class: Edge

Core. Edge



new Edge()

This class represents an edge in a Mesh.

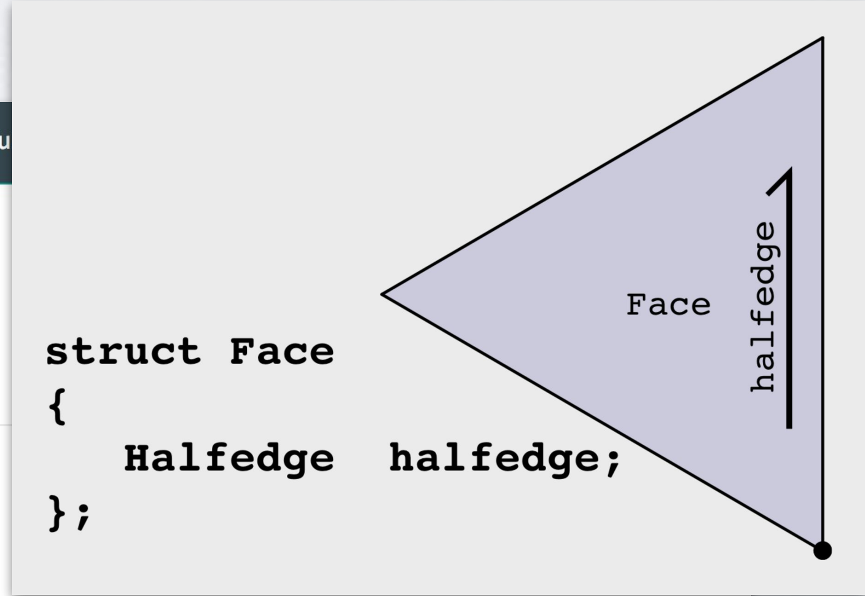
Properties:

Name	Type	Description
halfedge	module:Core.Halfedge	One of the halfedges associated with this edge.

geometry-processing-js Modu

Class: Face

Core. Face



new Face()

This class represents a face in a Mesh.

Properties:

Name	Type	Description
halfedge	module:Core.Halfedge	One of the halfedges associated with this face.

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

```
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

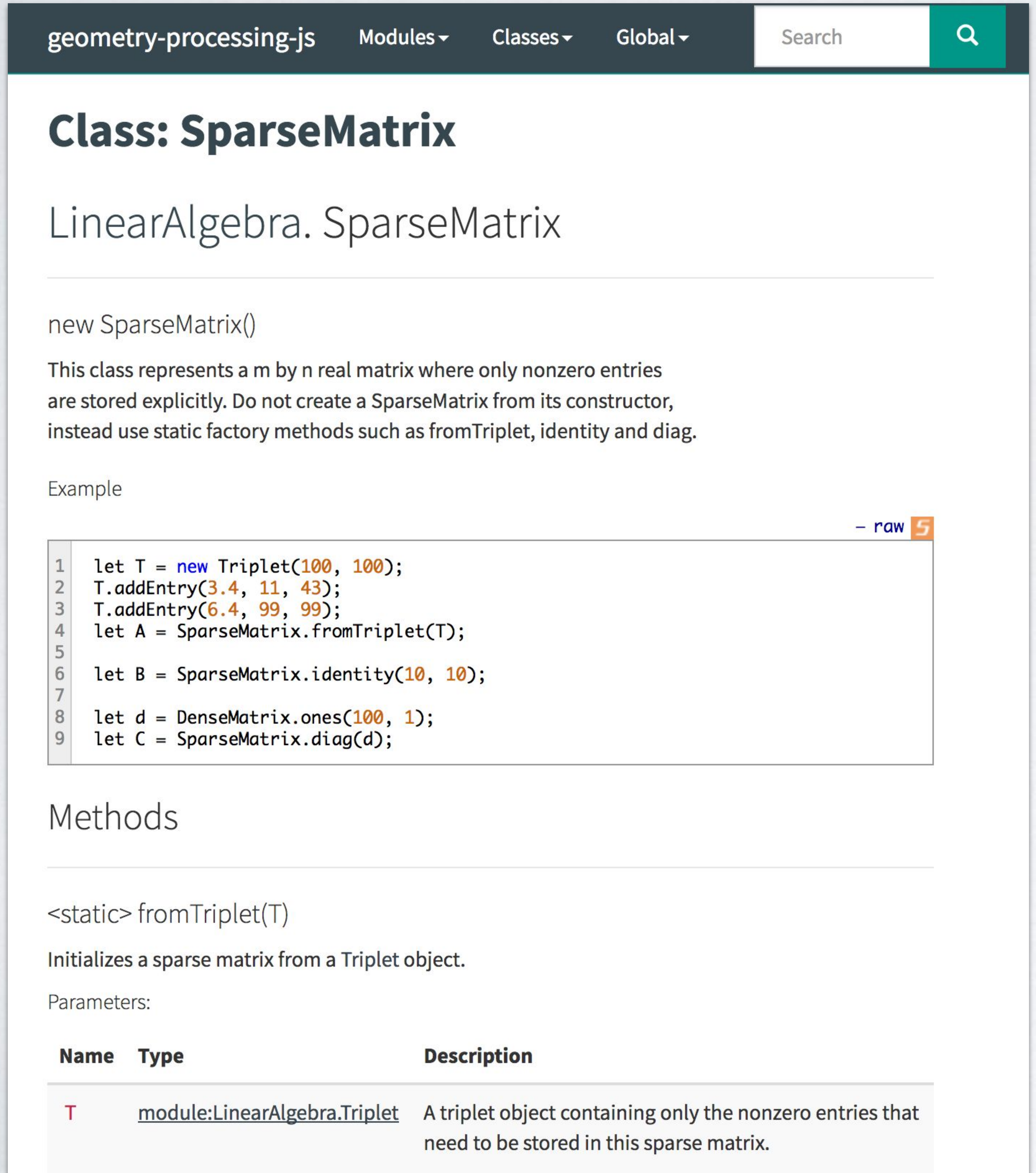
– raw

```
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

- Build from Triplet
- Modified version of CSC/CSR
- Eigen



geometry-processing-js Modules Classes Global Search

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

```
let T = new Triplet(100, 100);
T.addEntry(3.4, 11, 43);
T.addEntry(6.4, 99, 99);
let A = SparseMatrix.fromTriplet(T);

let B = SparseMatrix.identity(10, 10);

let d = DenseMatrix.ones(100, 1);
let C = SparseMatrix.diag(d);
```

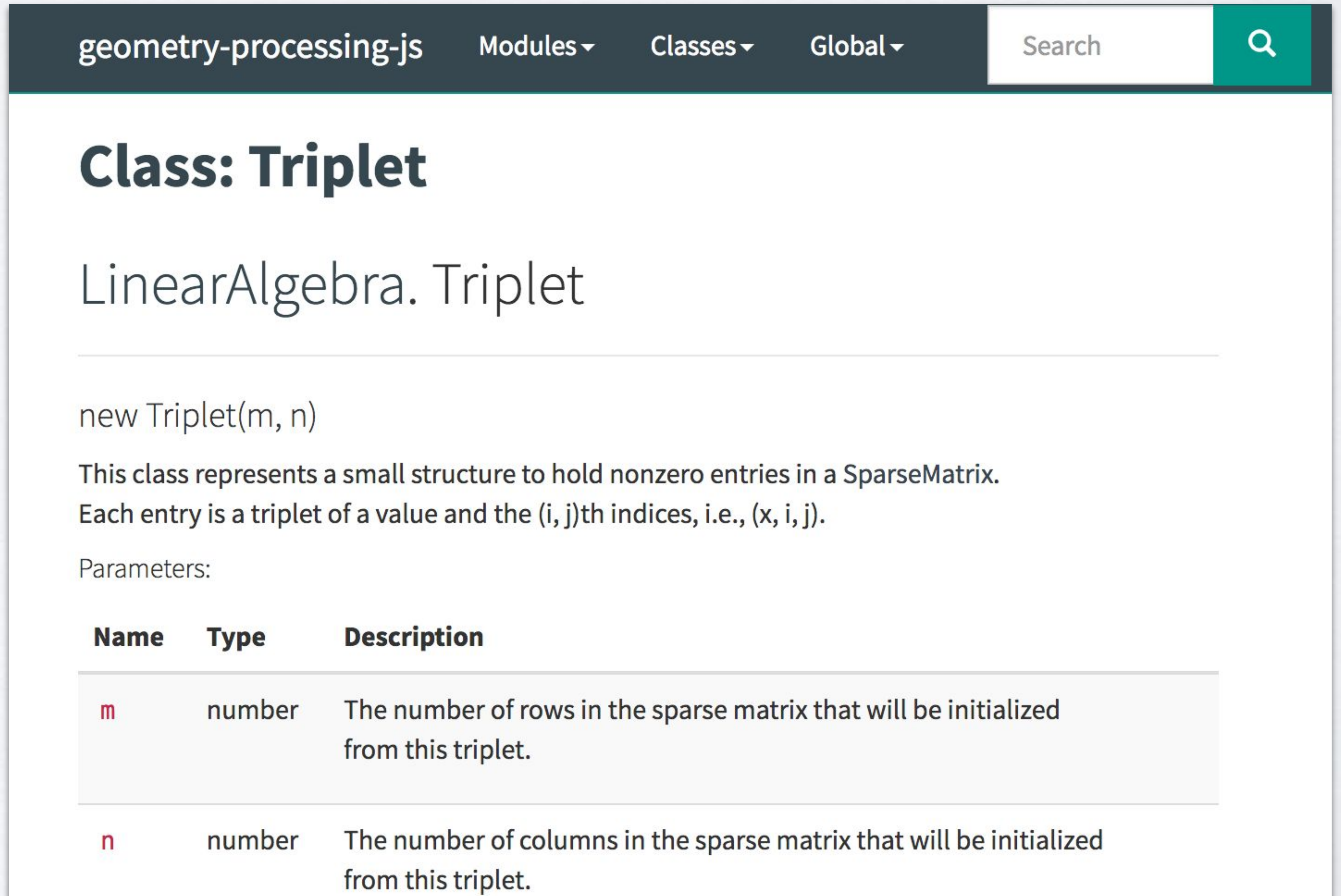
Methods

<static> fromTriplet(T)

Initializes a sparse matrix from a Triplet object.

Parameters:

Name	Type	Description
T	module:LinearAlgebra.Triplet	A triplet object containing only the nonzero entries that need to be stored in this sparse matrix.



geometry-processing-js Modules Classes Global Search

Class: Triplet

LinearAlgebra. Triplet

new Triplet(m, n)

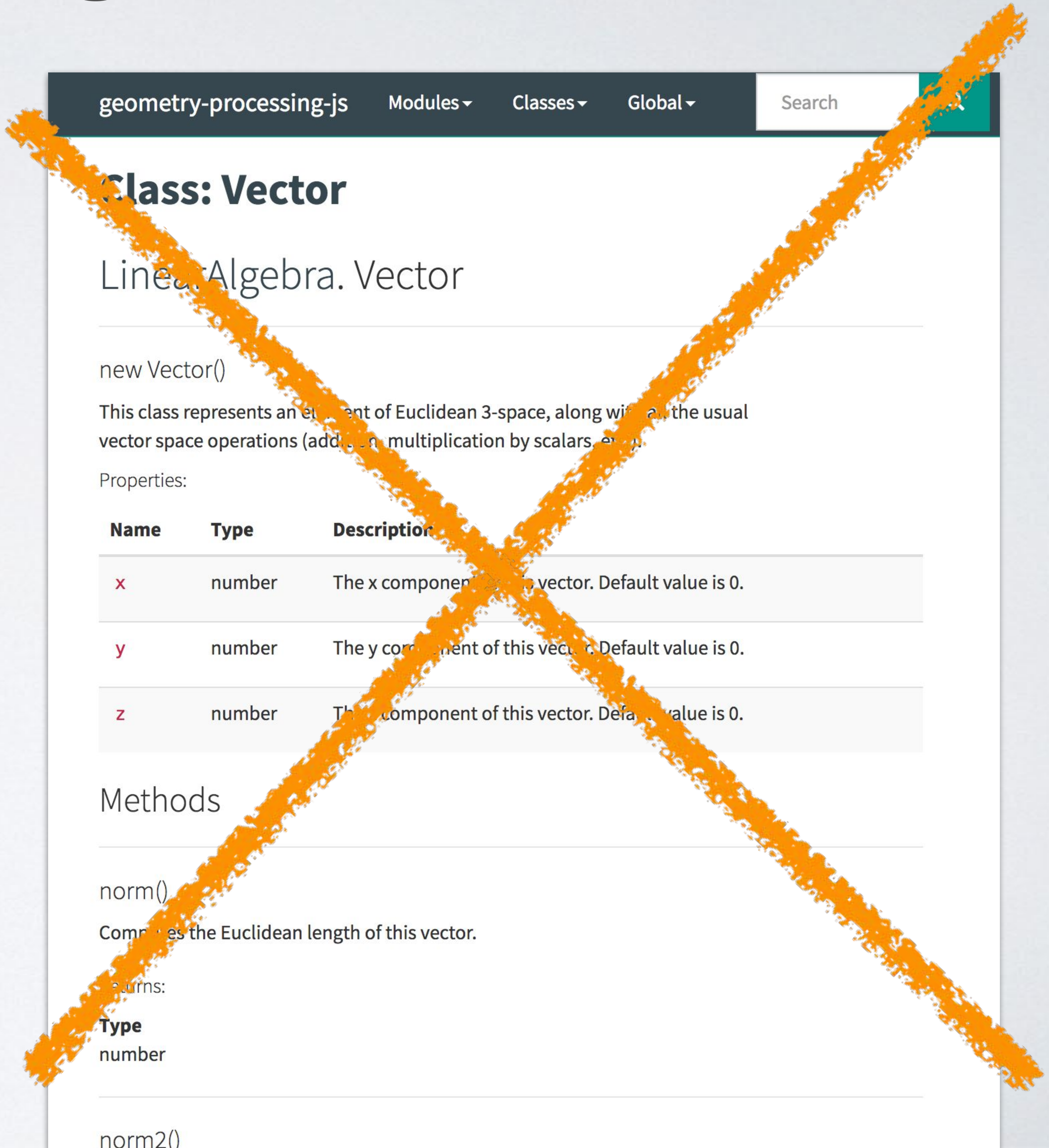
This class represents a small structure to hold nonzero entries in a SparseMatrix. Each entry is a triplet of a value and the (i, j)th indices, i.e., (x, i, j).

Parameters:

Name	Type	Description
m	number	The number of rows in the sparse matrix that will be initialized from this triplet.
n	number	The number of columns in the sparse matrix that will be initialized from this triplet.

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`

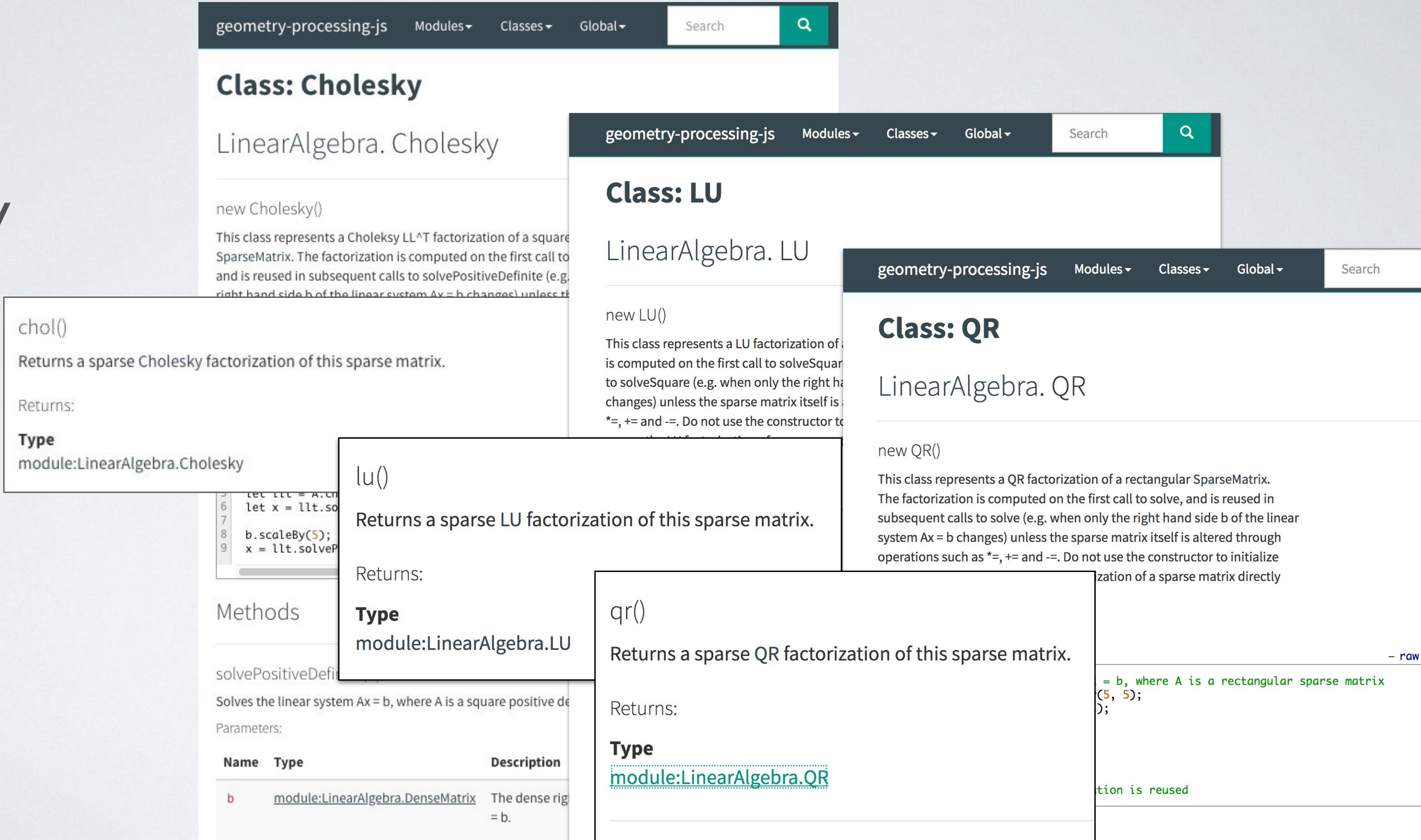


Solving linear systems

Cholesky

LU

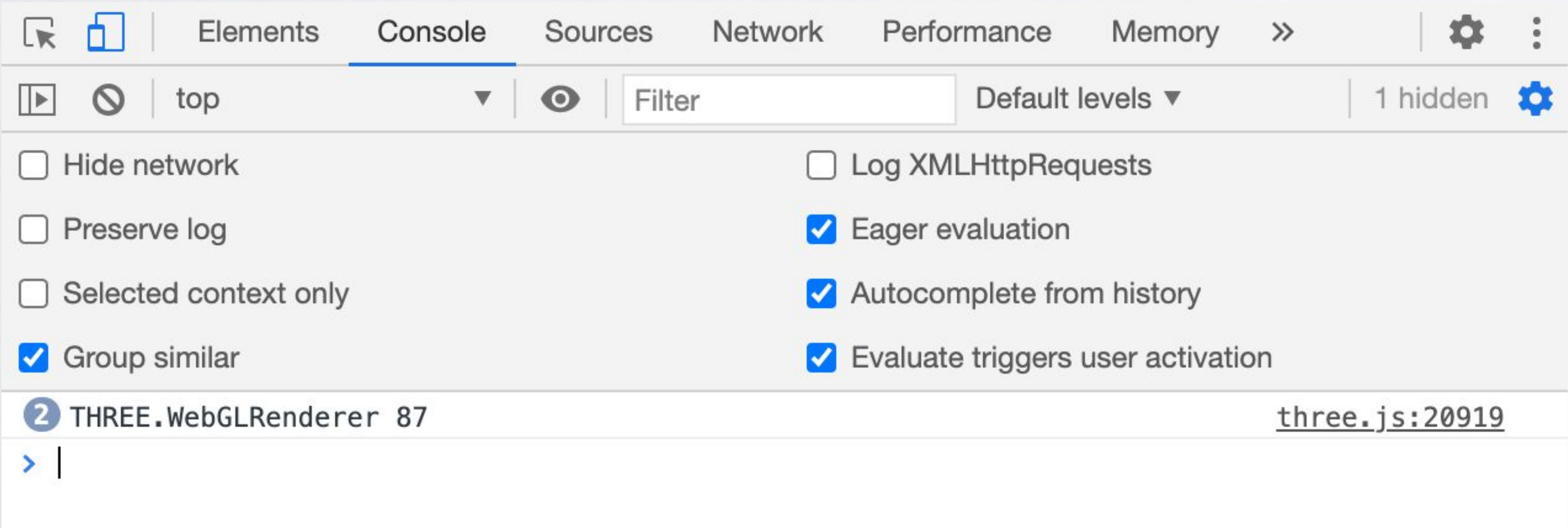
QR

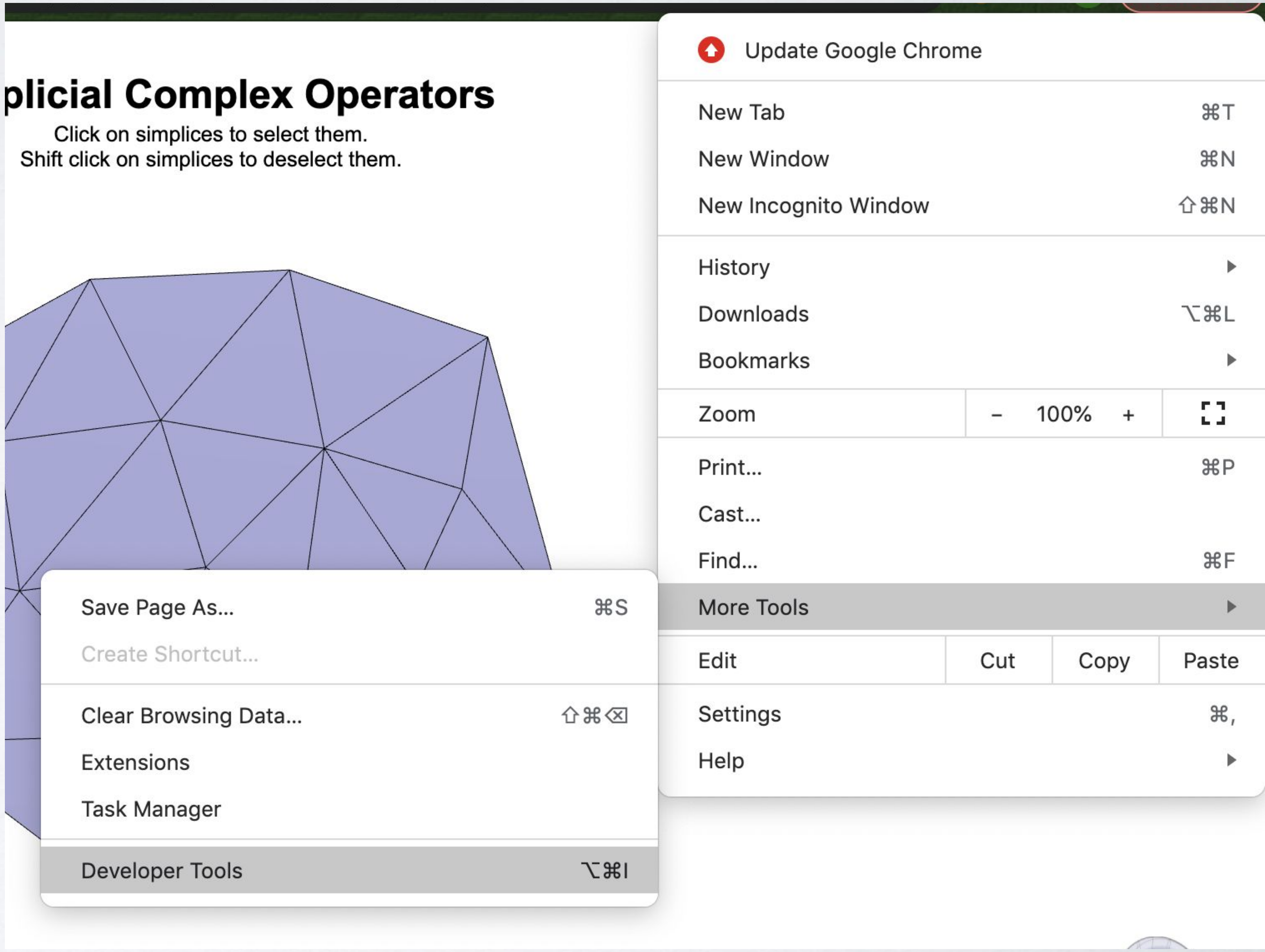


Print statements

Print using `console.log()`

Console is usually under "Developer tools" - might be different in your browser

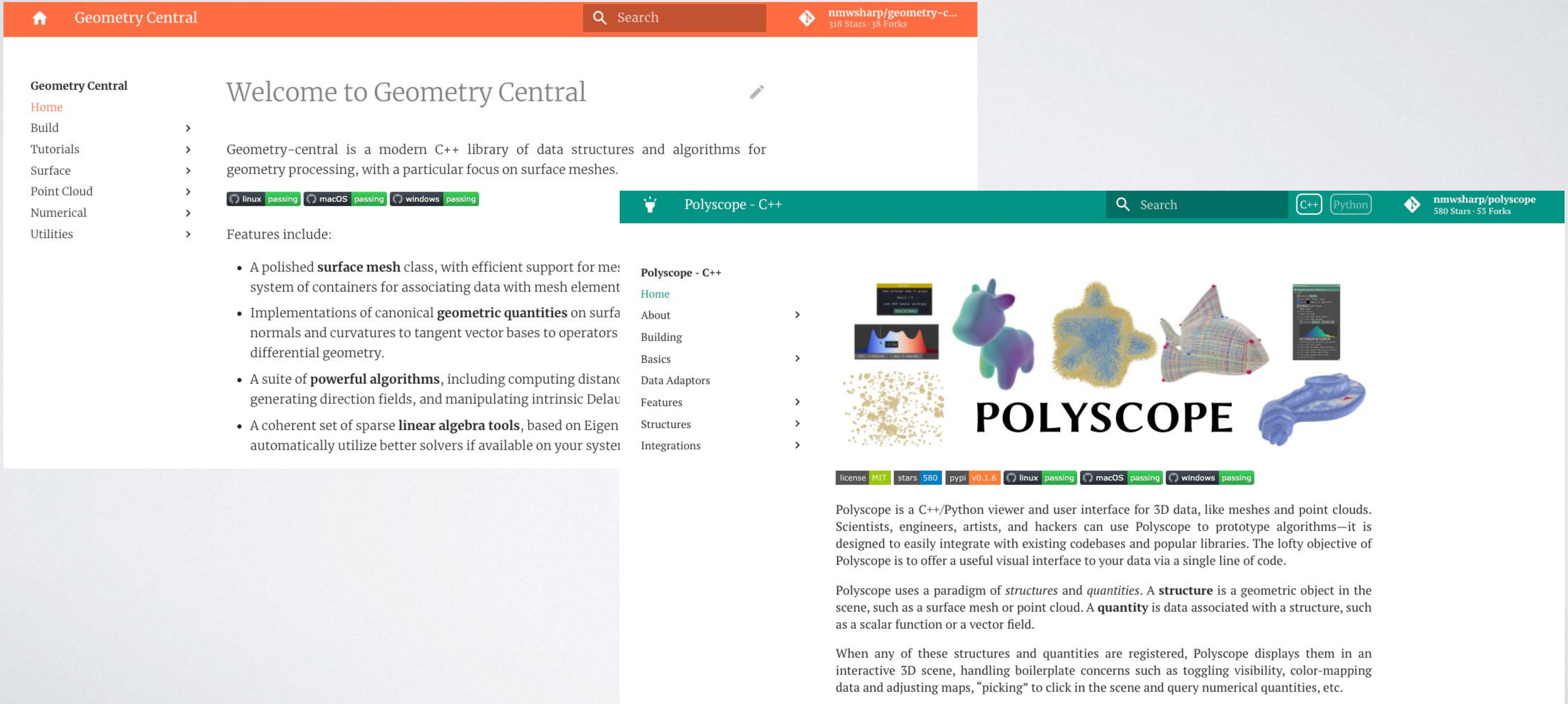




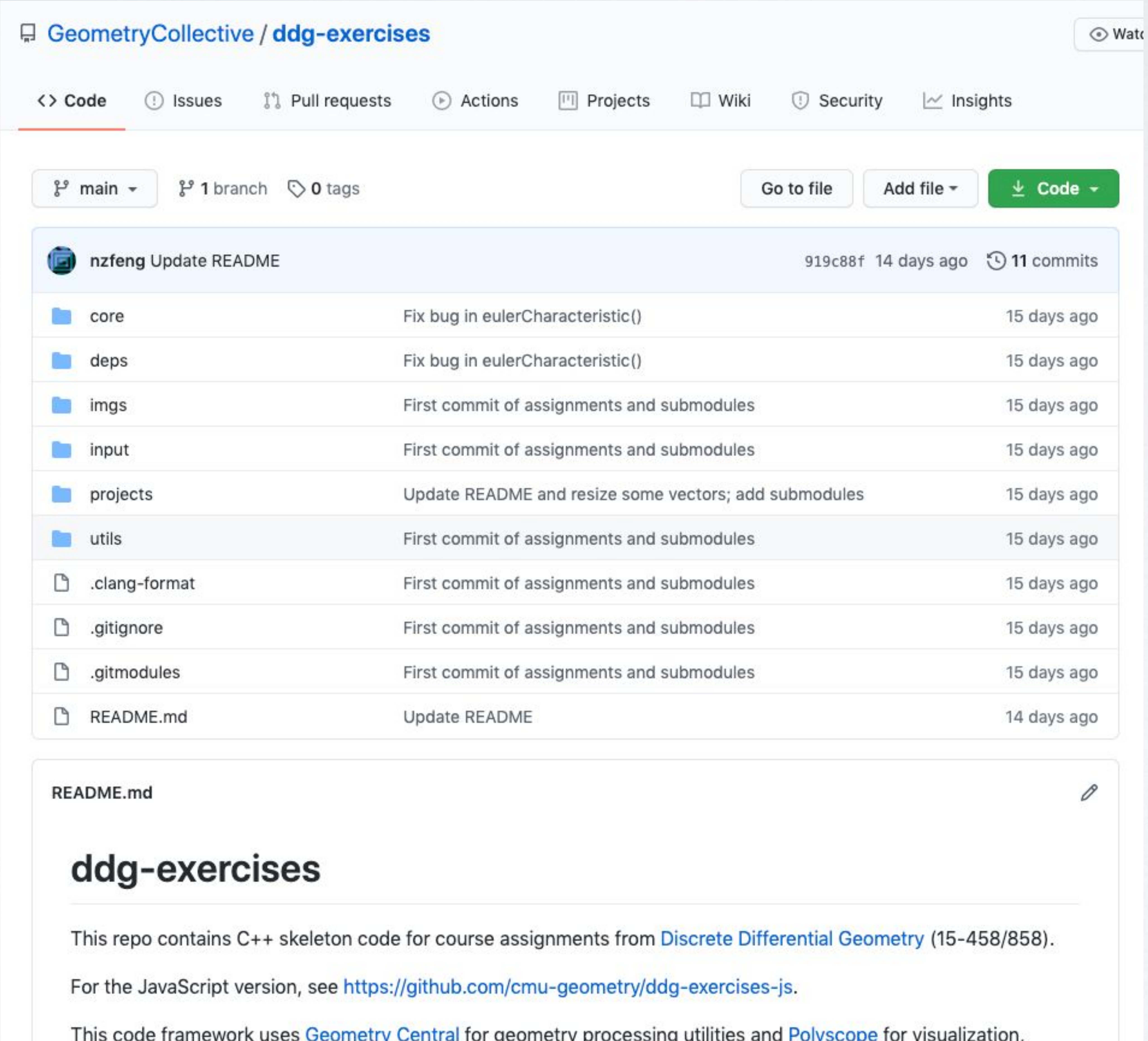
ddg-exercises (C++)

ddg-exercises

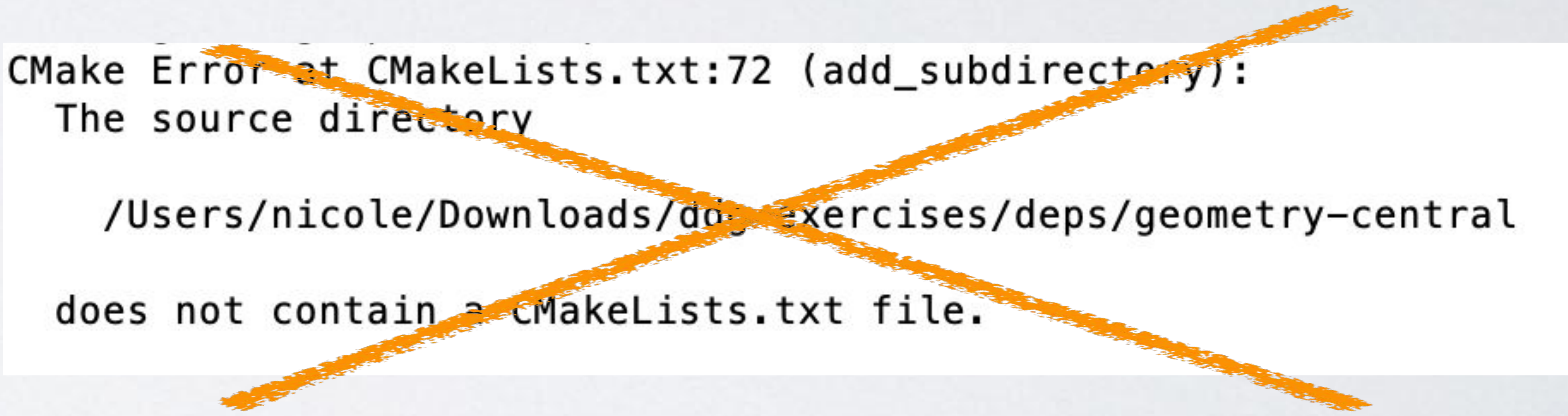
Uses Geometry Central and Polyscope (C++)



ddg-exercises



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



ddg-exercises

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 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
cmake ..
make
```

This builds an executable `main` which can then be run using

```
bin/main <optional_path_to_a_mesh>
```

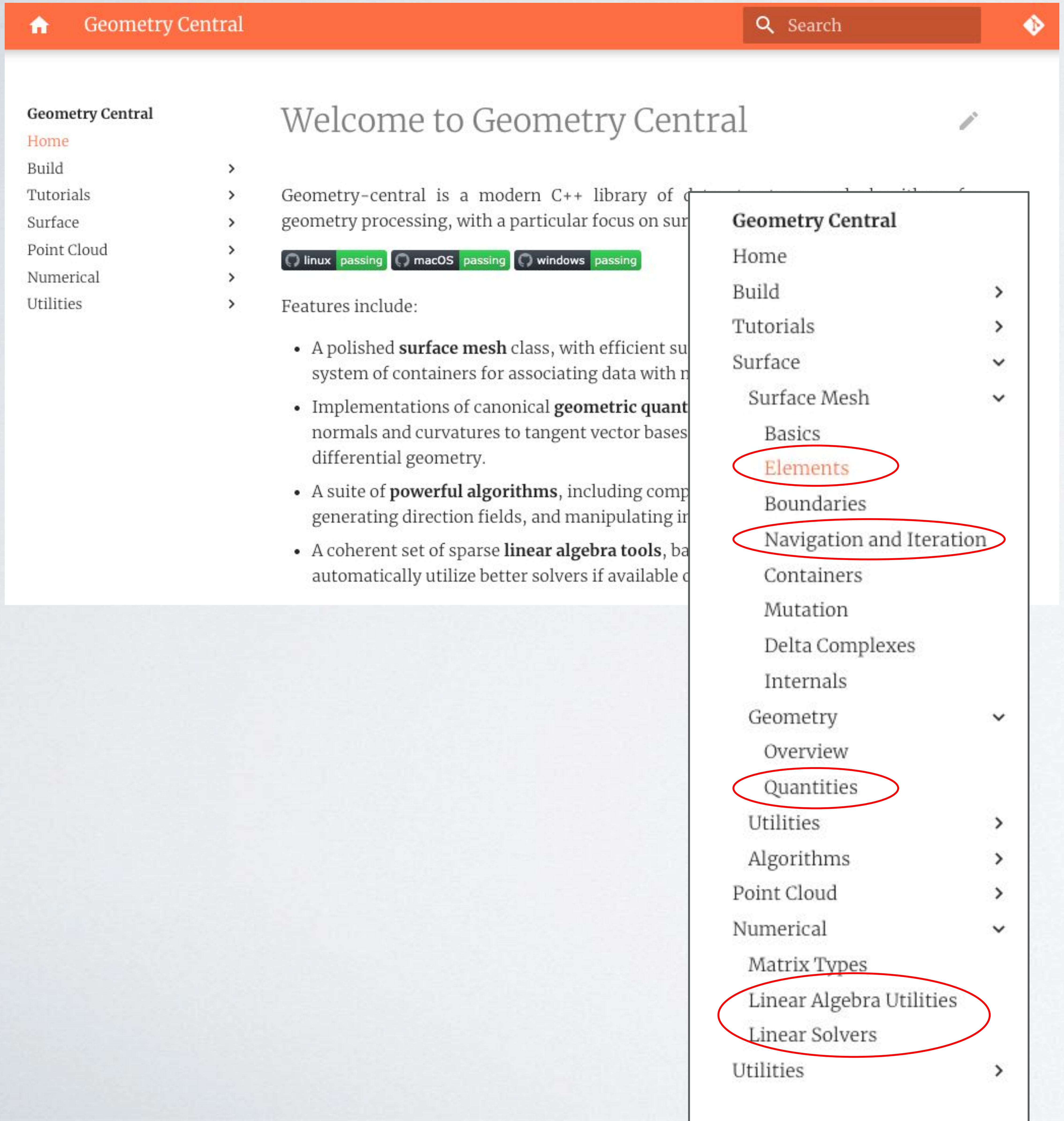
(See Geometry Central: Building for additional compiler flag options.

- 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

```
Nicoles-MacBook-Pro:build nicole$ bin/test-sco
[==========] Running 11 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 11 tests from SimplicialComplexOperatorsTest
[ RUN      ] SimplicialComplexOperatorsTest.isComplex
Testing isComplex()...
         A vertex:
/Users/nicole/Downloads/ddg-exercises/projects/simplicial-complex-operators/src/test-sco.cpp:49: Failure
Expected equality of these values:
  expectedResult
    Which is: 1
  SCO.isComplex(S)
    Which is: false
                 isComplex() returns wrong result for a vertex
```

......

```
Value of: edgesAreCorrect
  Actual: false
Expected: true
                 The edges in your link are wrong
         Link of a face:
/Users/nicole/Downloads/ddg-exercises/projects/simplicial-complex-operators/src/test-sco.cpp:86: Failure
Value of: facesAreCorrect
  Actual: false
Expected: true
                 The faces in your link are wrong
[  FAILED  ] SimplicialComplexOperatorsTest.link (240 ms)
[----------] 11 tests from SimplicialComplexOperatorsTest (3066 ms total)

[----------] Global test environment tear-down
[==========] 11 tests from 1 test suite ran. (3066 ms total)
[  PASSED  ] 0 tests.
[  FAILED  ] 11 tests, listed below:
[  FAILED  ] SimplicialComplexOperatorsTest.isComplex
[  FAILED  ] SimplicialComplexOperatorsTest.isPureComplex
[  FAILED  ] SimplicialComplexOperatorsTest.A0
[  FAILED  ] SimplicialComplexOperatorsTest.A1
[  FAILED  ] SimplicialComplexOperatorsTest.buildVertexVector
[  FAILED  ] SimplicialComplexOperatorsTest.buildEdgeVector
[  FAILED  ] SimplicialComplexOperatorsTest.buildFaceVector
[  FAILED  ] SimplicialComplexOperatorsTest.boundary
[  FAILED  ] SimplicialComplexOperatorsTest.star
[  FAILED  ] SimplicialComplexOperatorsTest.closure
[  FAILED  ] SimplicialComplexOperatorsTest.link

11 FAILED 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

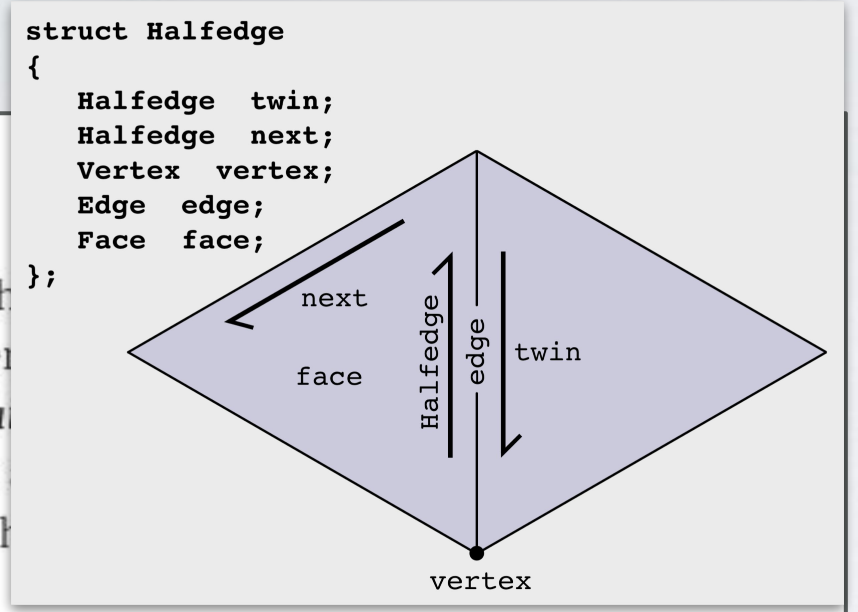
- Write code in project folder or `core/`, in one or more of the source (.cpp) files
- We've handled visualization in Polyscope
- Generate Makefile using cmake
- `make` and `bin/main` to run program!
- Additional meshes provided in `inputs/` (up a few directories relative to `projects/`)

Navigating halfedges

In Geometry Central

Halfedge

A halfedge is a the basic building block of a h
halfedge is *half* of an *edge*, connecting two ve
some face. The halfedge is directed, from its *ta*
clockwise orientation: the halfedges with in
clockwise direction. On a `ManifoldSurfaceMesh` , a h
an edge) will point in opposite directions.



Traversal:

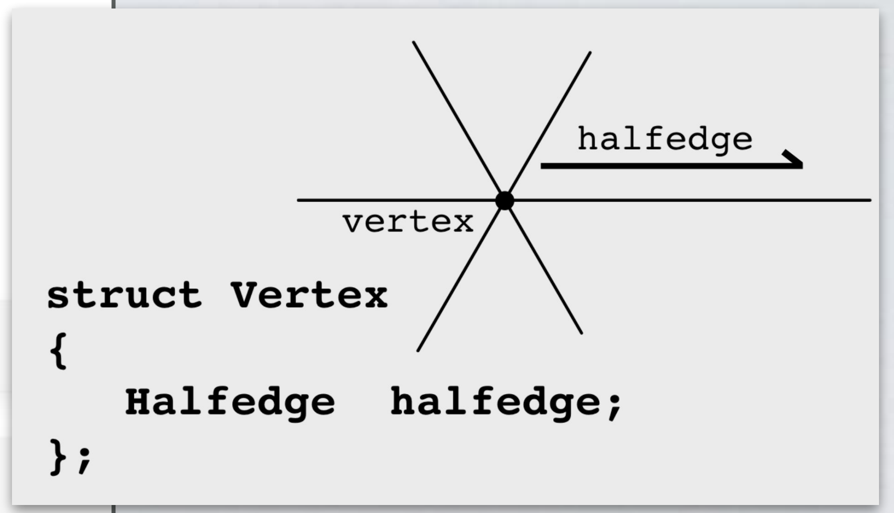
- `Halfedge Halfedge::twin()`
- `Halfedge Halfedge::sibling()`
- `Halfedge Halfedge::next()`
- `Vertex Halfedge::vertex()`
- `Vertex Halfedge::tailVertex()`
- `Vertex Halfedge::tipVertex()`
- `Edge Halfedge::edge()`
- `Face Halfedge::face()`
- `Corner Halfedge::corner()`

Vertex

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

Traversal:

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

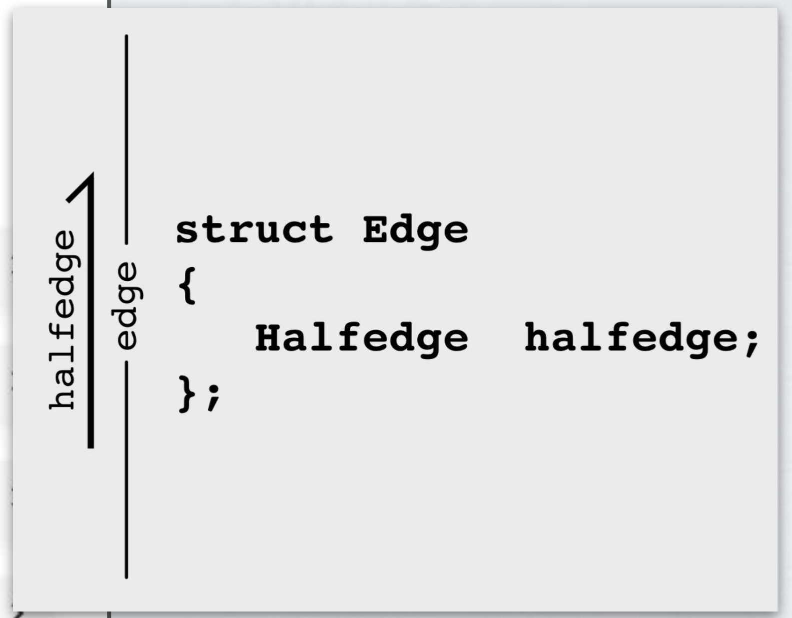


Edge

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

Traversal:

- `Halfedge Edge::halfedge()`
- `Vertex Edge::otherVertex(Vertex v)`
- `Vertex Edge::firstVertex()`
- `Vertex Edge::secondVertex()`

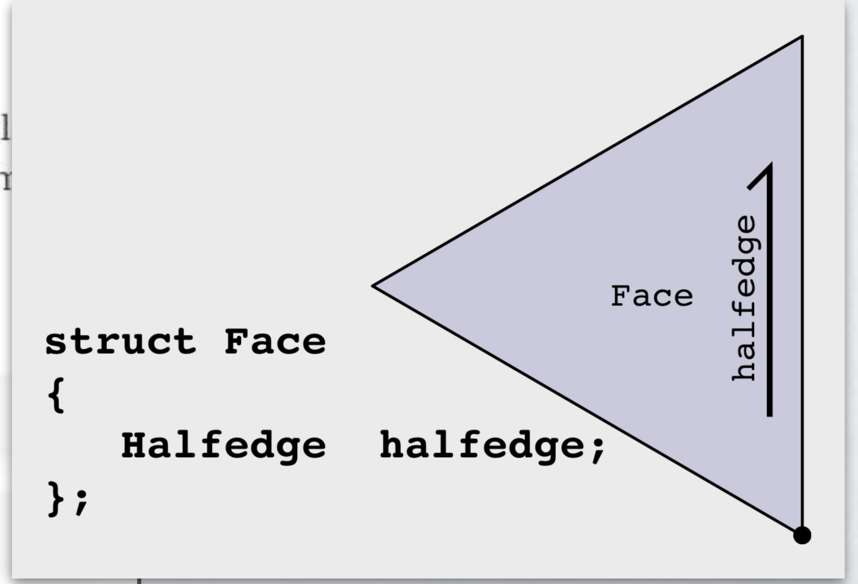


Face

A *face* is a 2-dimensional element formed by a loop of 3 or more edges. In general
faces can be polygonal with $d \geq 3$ edges, though many of the routines in geom
central are only valid on triangular meshes.

Traversal:

- `Halfedge Face::halfedge()`
- `BoundaryLoop Face::asBoundaryLoop()`



In Geometry Central

Includes many convenience functions

(see Navigation and Iteration documentation)

Around a vertex

Vertex::outgoingHalfedges()

Iterate over the halfedges which point outward from a vertex.

```
for(Halfedge he : vert.outgoingHalfedges()) {
  assert(he.vertex() == vert); // true
  // do science here
}
```

Vertex::incomingHalfedges()

Vertex::adjacentVertices()

Vertex::adjacentEdges()

Vertex::adjacentFaces()

Around an edge

Edge::adjacentHalfedges()

Edge::adjacentFaces()

Edge::adjacentVertices()

Edge::diamondBoundary()

Around a face

Face::adjacentVertices()

Face::adjacentHalfedges()

Face::adjacentEdges()

Face::adjacentFaces()

Linear algebra in Geometry Central

Sparse Matrices in Geometry Central

Linear algebra utilities

Construct and convert

`SparseMatrix<T> identityMatrix(size_t N)`

Construct and `N x N` identity matrix of the requested type.

`void shiftDiagonal(SparseMatrix<T>& m, T shiftAmount = 1e-4)`

Shift the diagonal of matrix, by adding `A + shiftDiagonal * identityMatrix()`.

- 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:

```
// Define a triplet that represents a matrix element of type double.
typedef Eigen::Triplet<double> T;
// A vector to store our triplets
std::vector<T> tripletList;
// Initialize a Geometry Central sparse matrix of size (nrows x ncols),
// and holds elements of type double
SparseMatrix<double> M(nrows, ncols);
// Add some nonzero elements to our matrix.
tripletList.push_back(T(row_idx1, col_idx1, val1));
tripletList.push_back(T(row_idx2, col_idx2, val2));
tripletList.push_back(T(row_idx3, col_idx3, val3));
// Set the matrix with the values we defined.
M.setFromTriplets(tripletList.begin(), tripletList.end());
```

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 Suitesprase solvers will be used intead. 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.

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

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

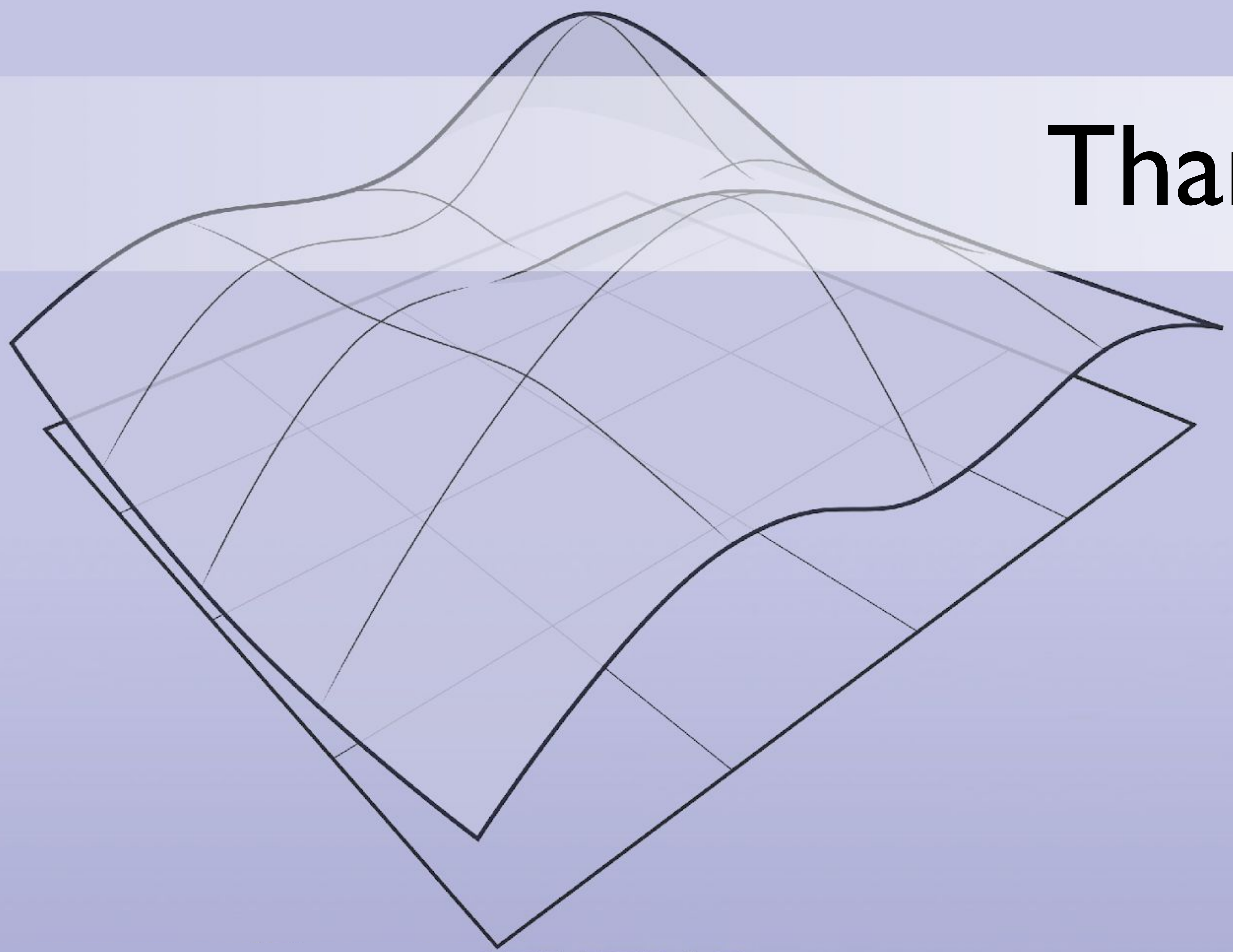
Solve a system with a *square* matrix. Uses an LU decomposition interally.

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

Solve a system with a *symmetric positive (semi-)definite* matrix. Uses an LDLT decomposition interally.

Geometry Central conveniently provides functions for solving square or SPD matrices, that use LU or Cholesky decomposition

Thanks!



Discrete Differential Geometry:
An Applied Introduction