

Discrete Differential Geometry: An Applied Introduction

Spring 2024 TAs: Oscar Chen and Olga Guţan

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



Outline

Outline

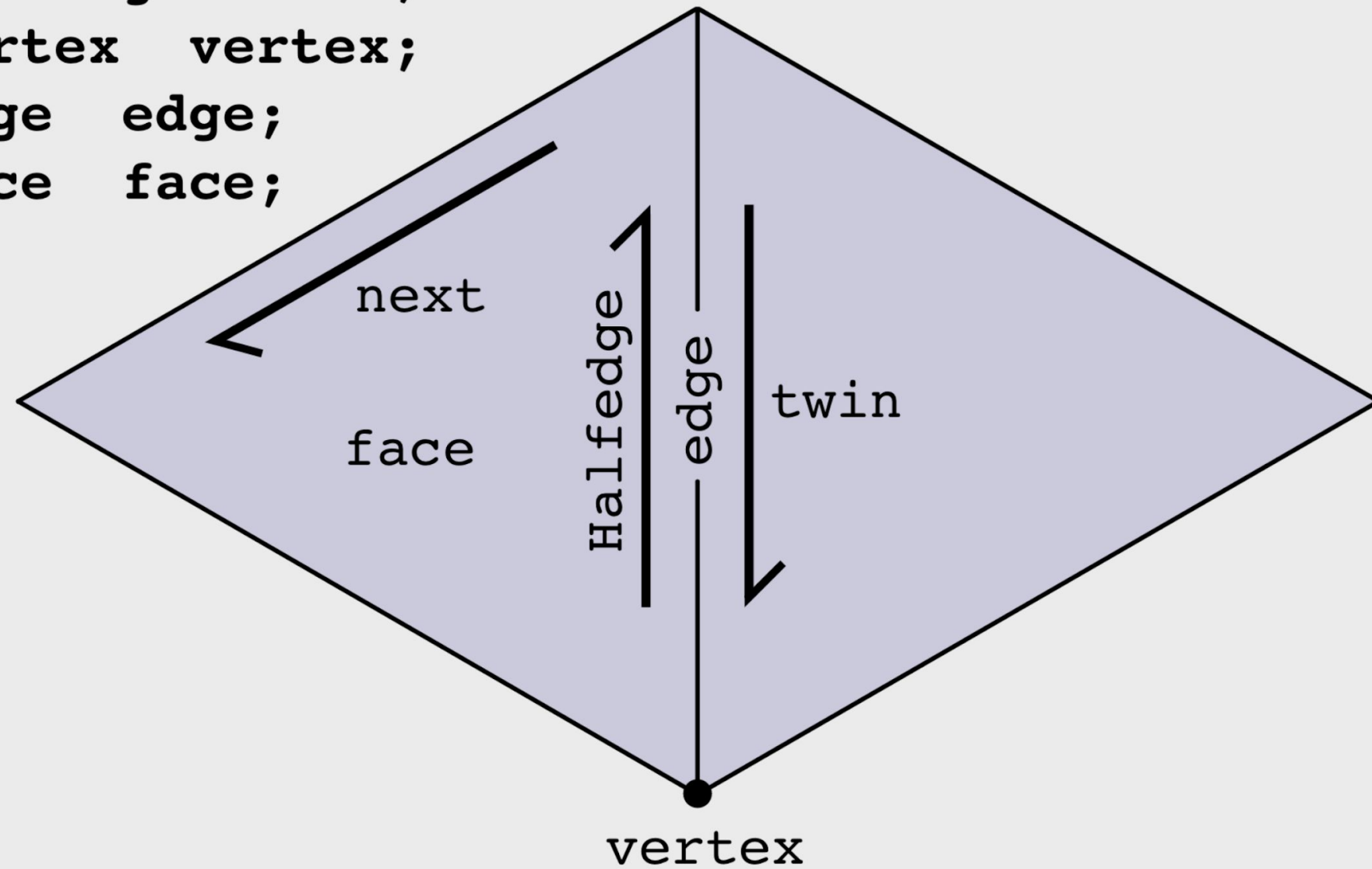
- Halfedge data structure
- Sparse matrices
- Solving linear systems (direct methods)
- Intro to either C++ or JS



The Halfedge Data Structure

The Halfedge Data Structure

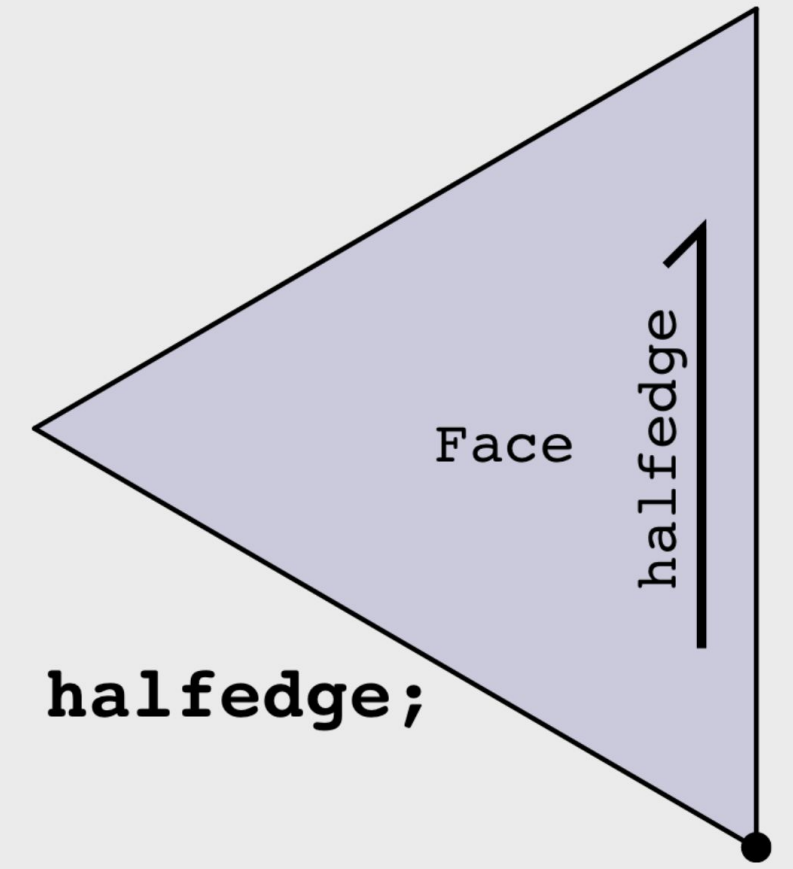
```
struct Halfedge
{
    Halfedge twin;
    Halfedge next;
    Vertex vertex;
    Edge edge;
    Face face;
};
```



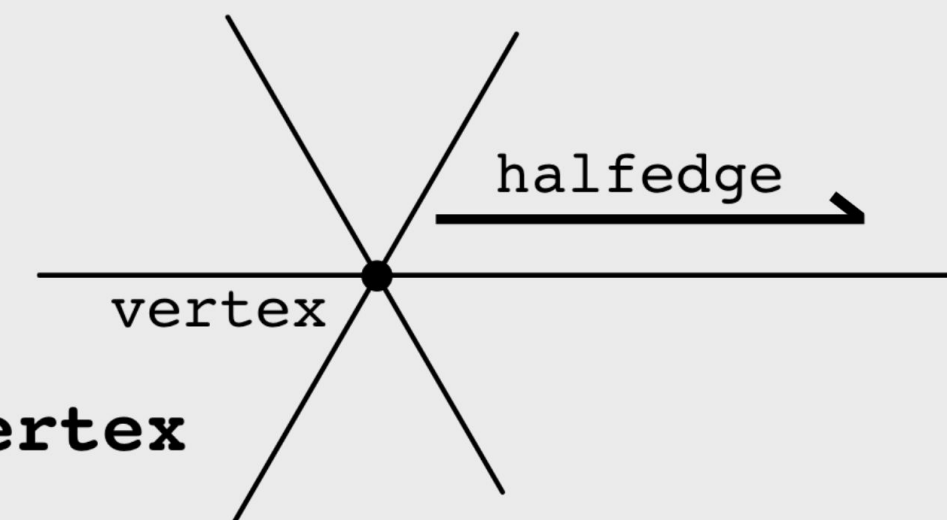
```
struct Edge
{
    Halfedge halfedge;
};
```

A diagram illustrating the edge structure. A vertical line segment is shown. A label 'edge' is placed to the left of the line, and a label 'halfedge' is placed to the right of the line, with an arrow pointing to the line.

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



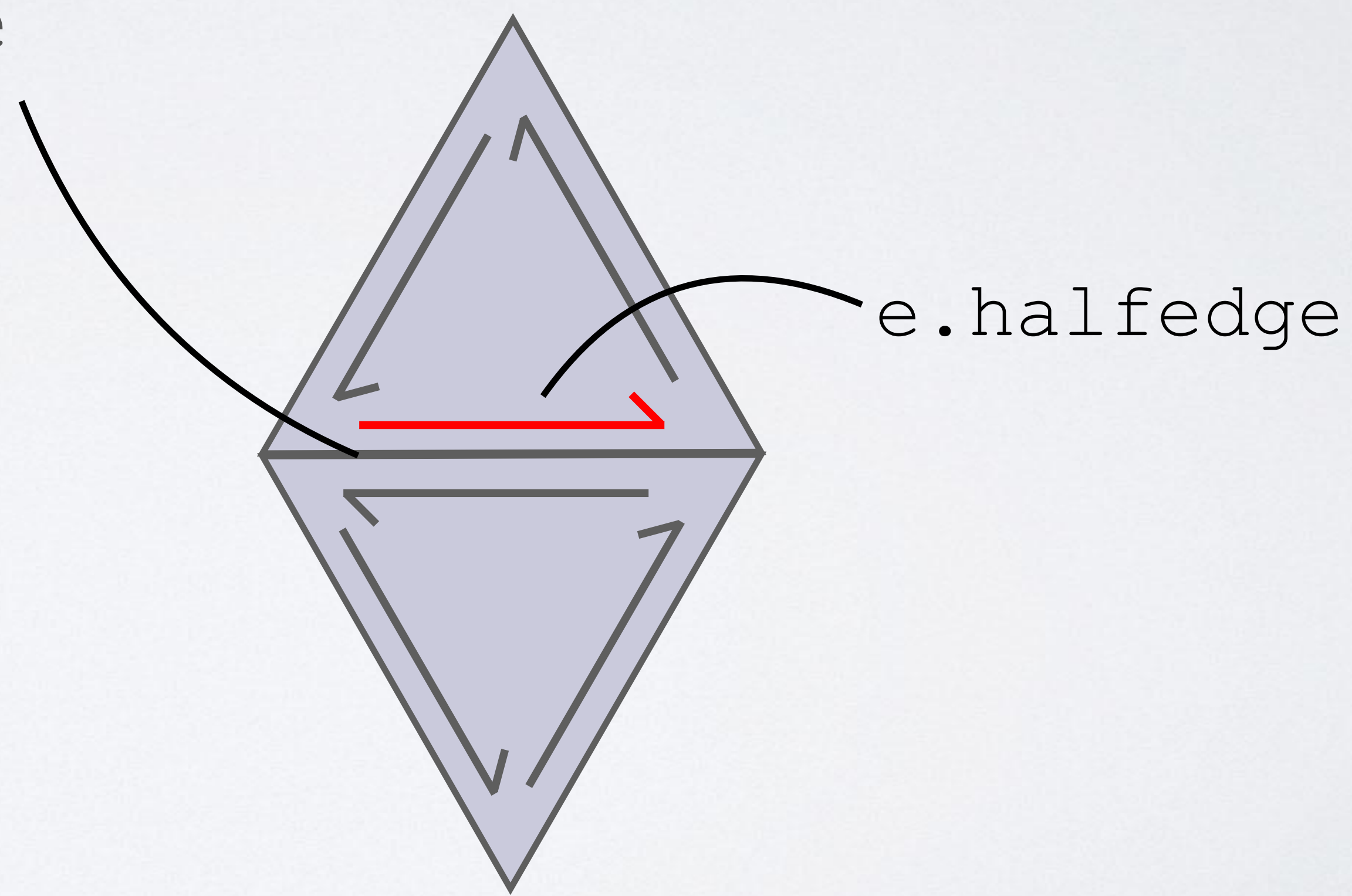
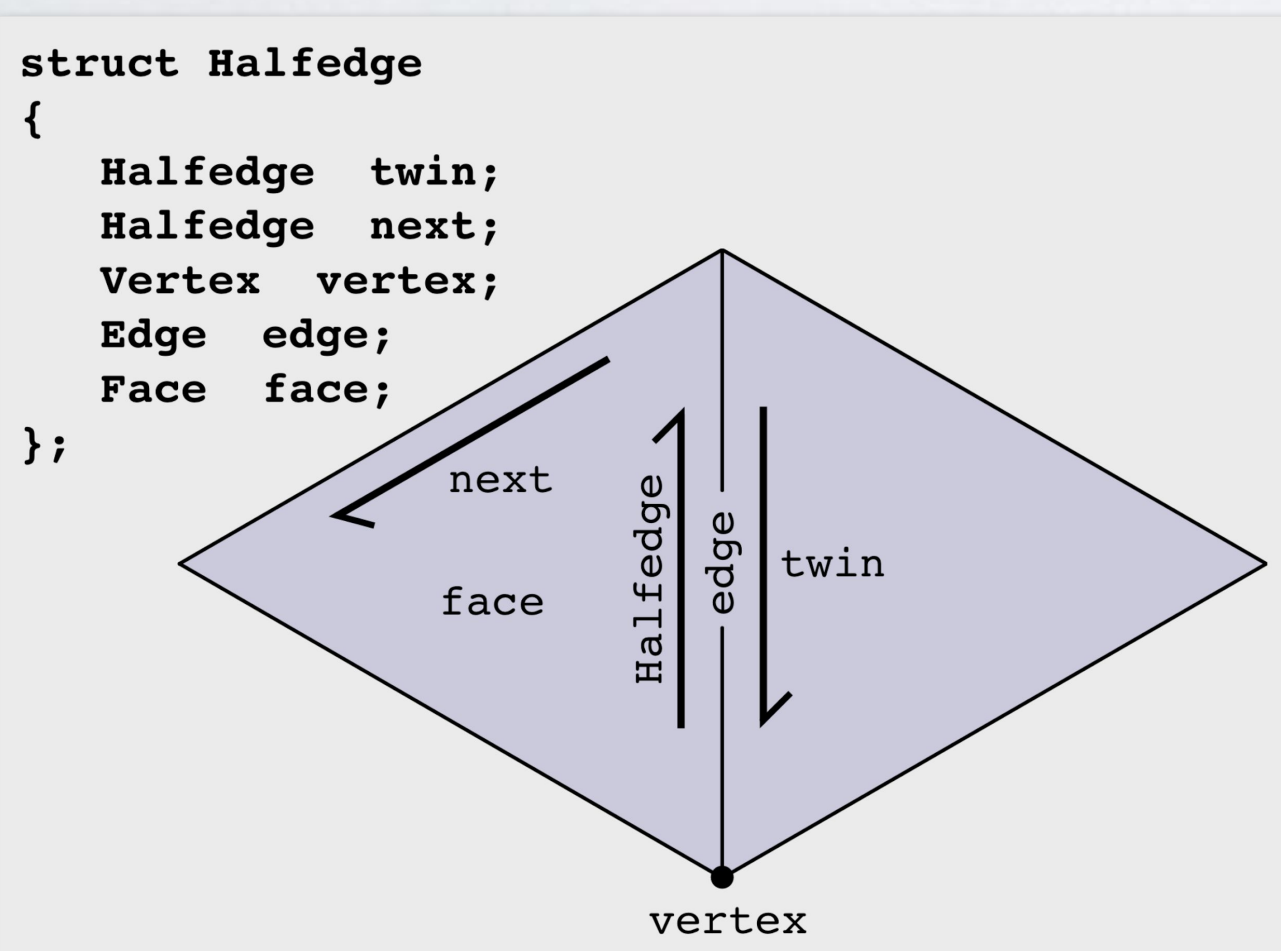
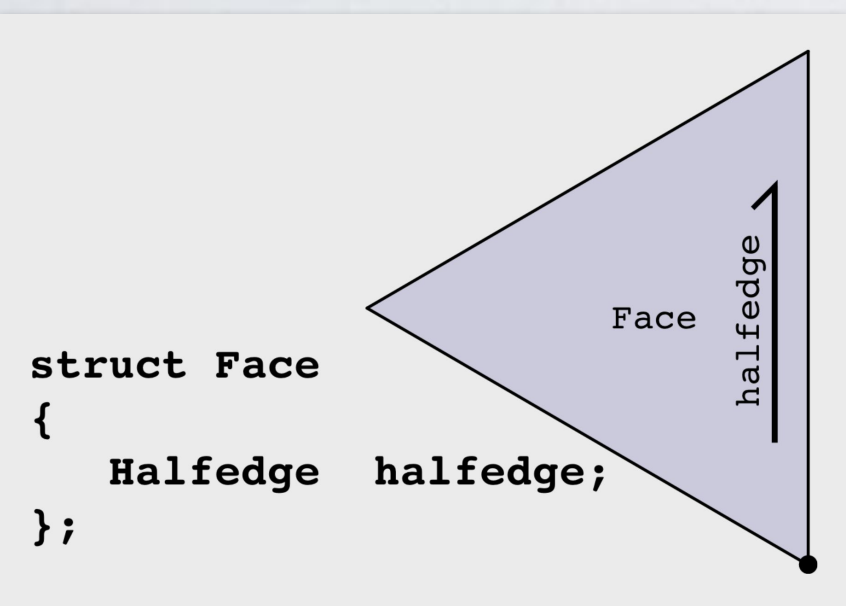
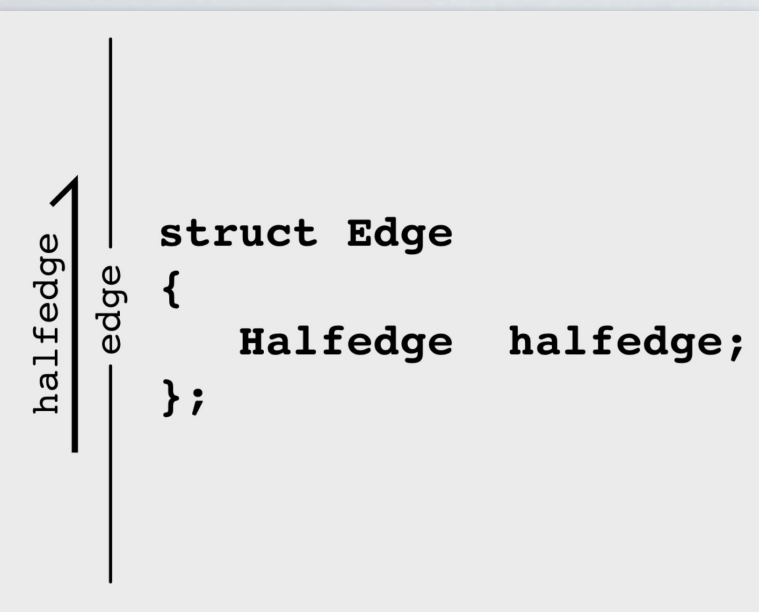
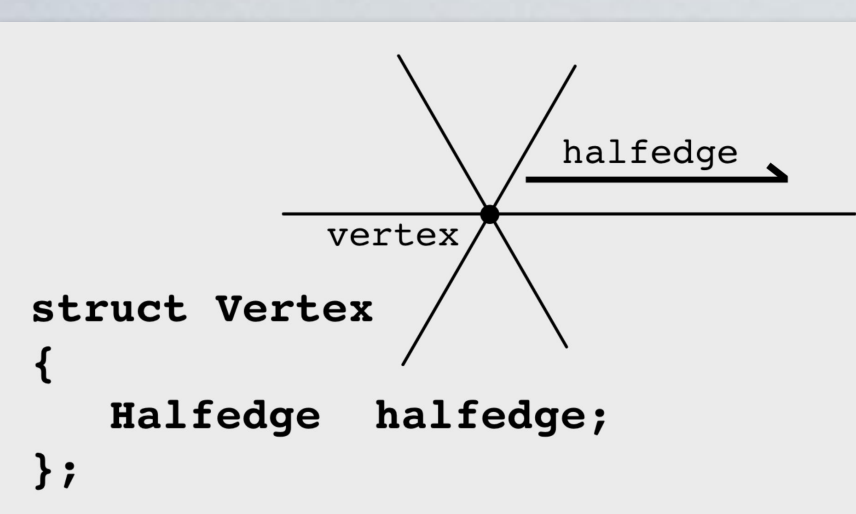
```
struct Vertex
{
    Halfedge halfedge;
};
```

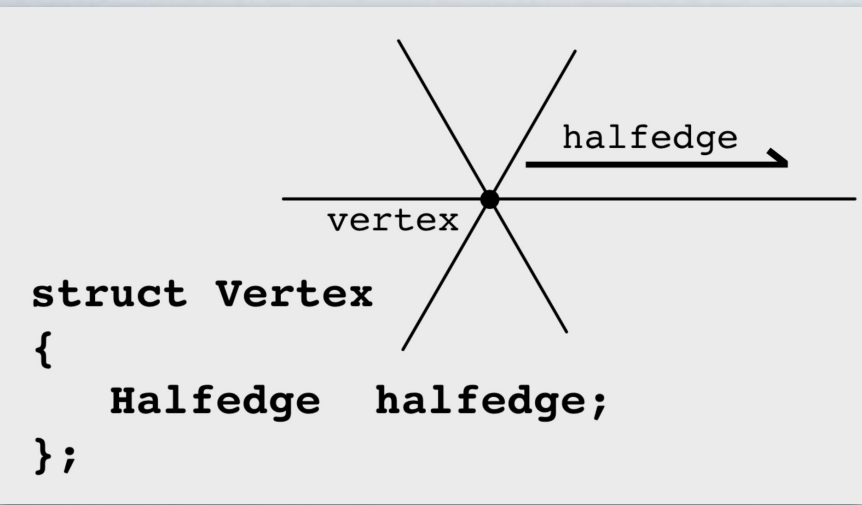


The Halfedge Data Structure

How would I find the faces adjacent to an edge?

Given: Edge e

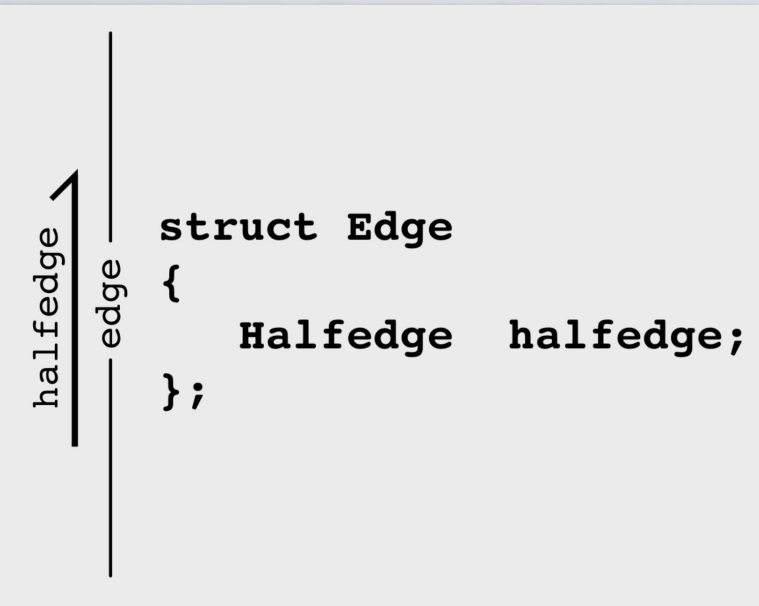




The Halfedge Data Structure

How would I find the faces adjacent to an edge?

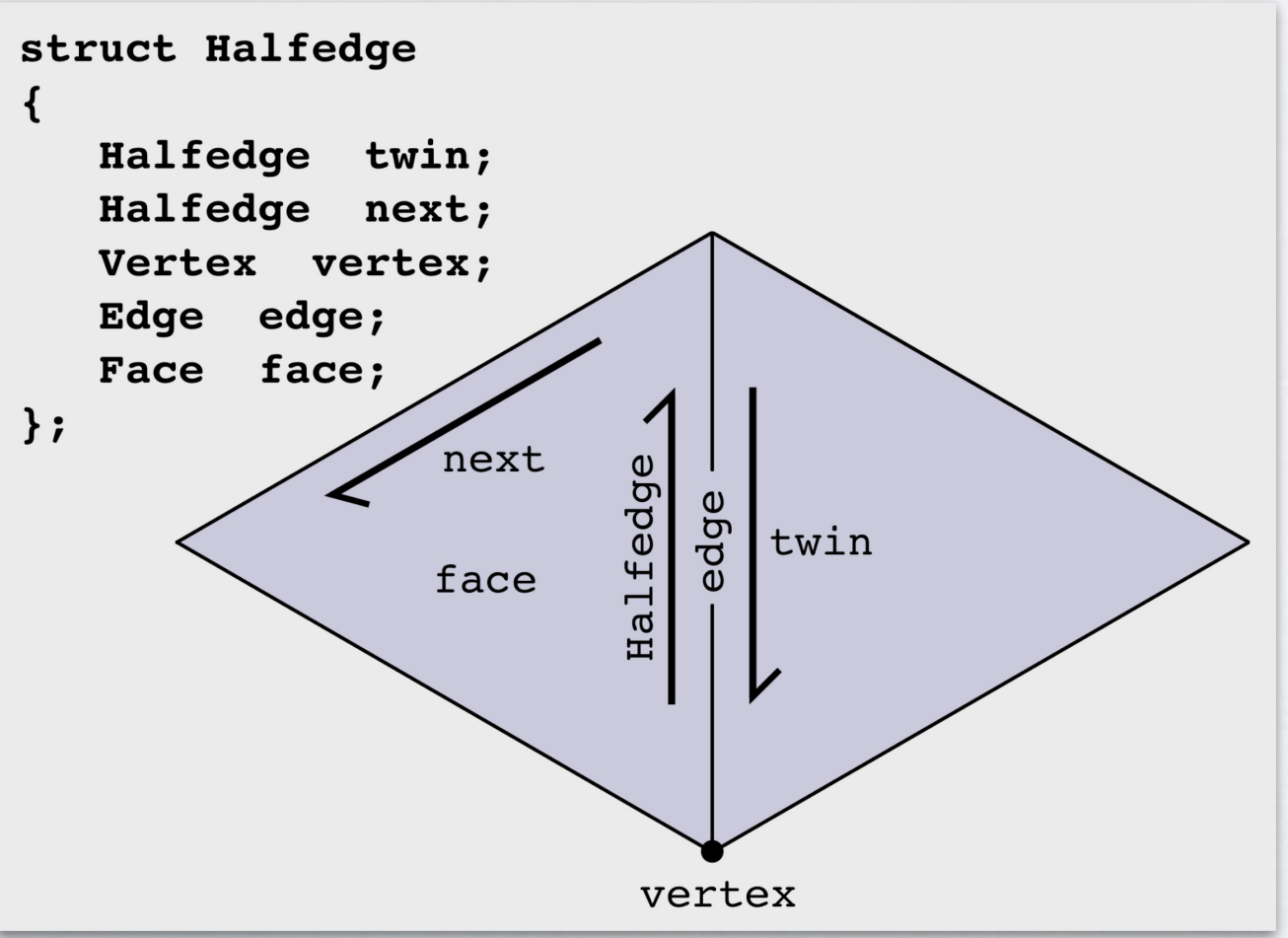
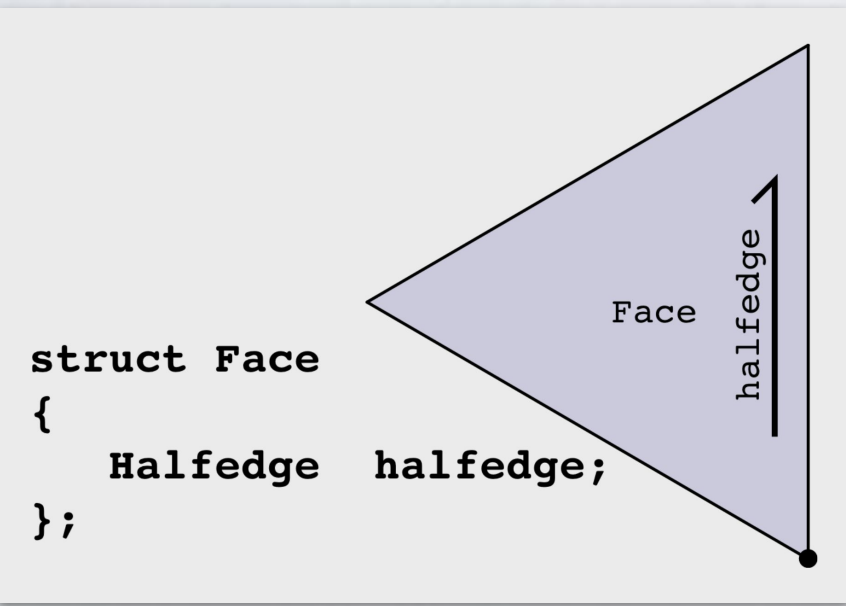
Given: Edge e



Halfedge he = e.halfedge;

Face left_face = he.face;

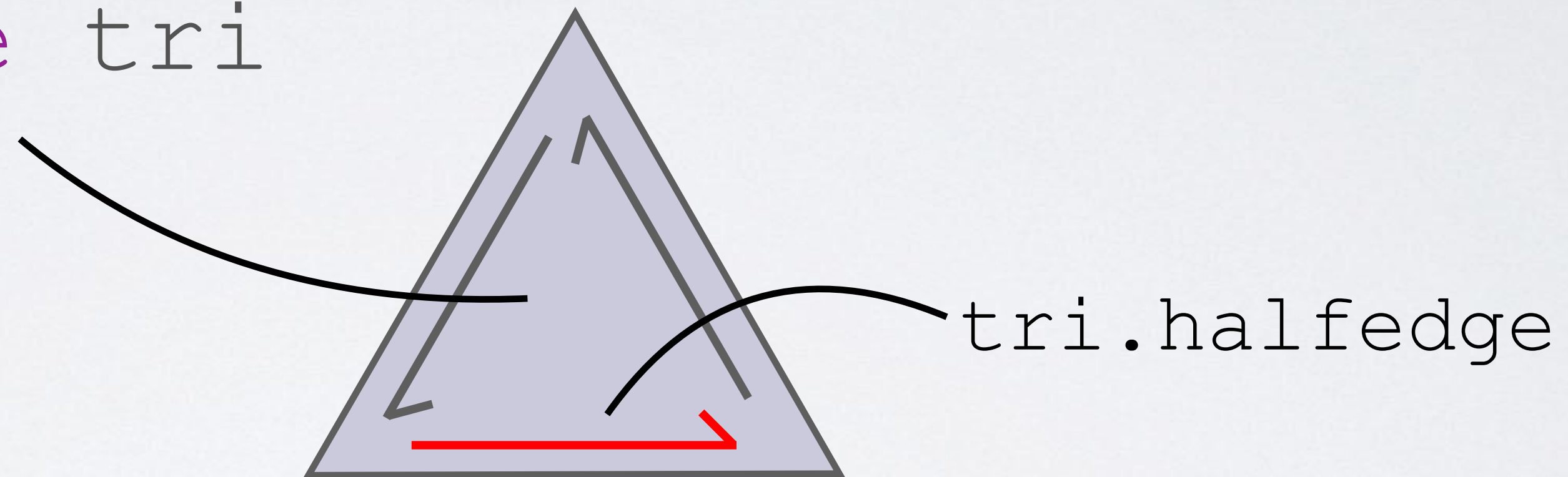
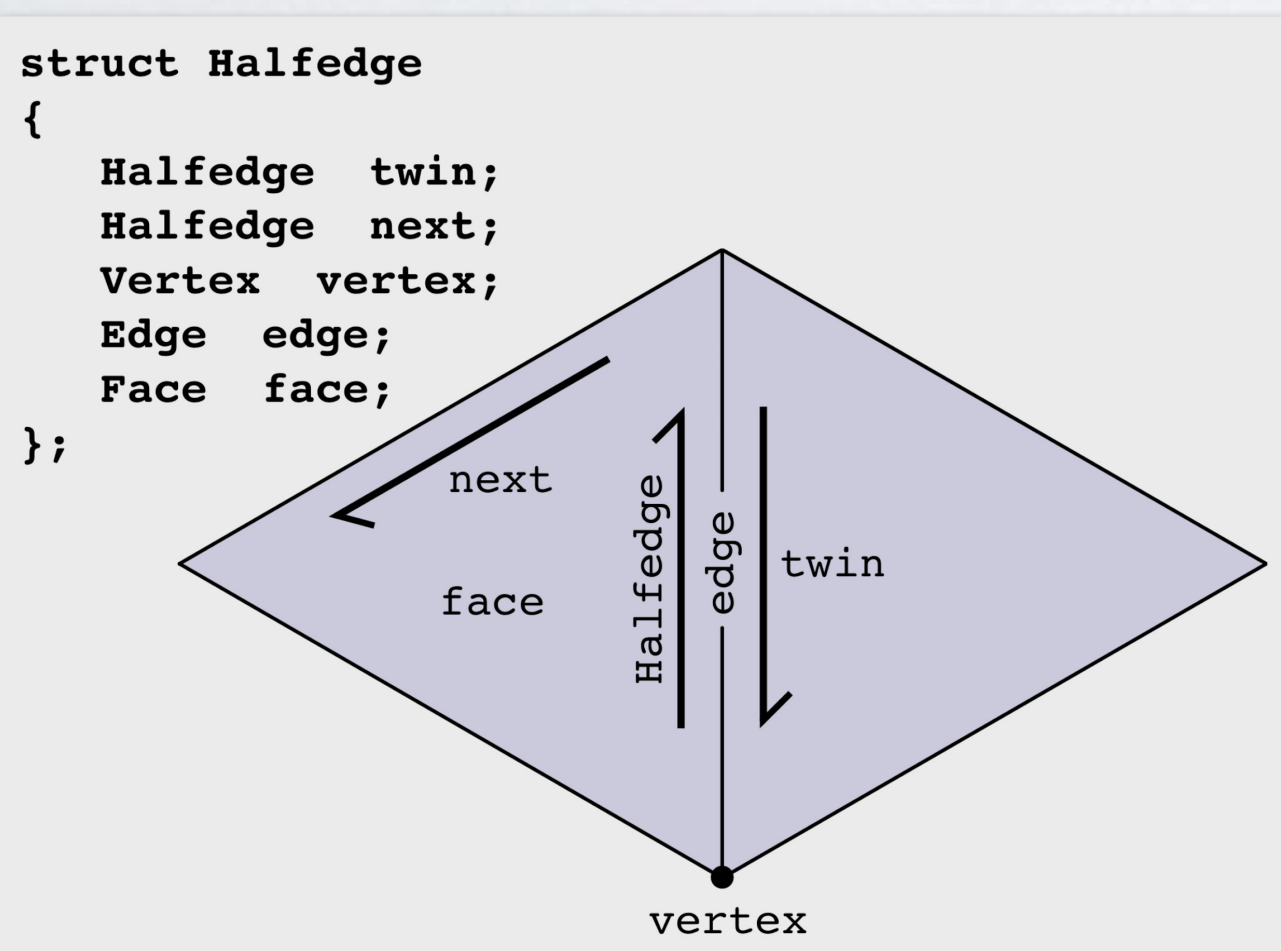
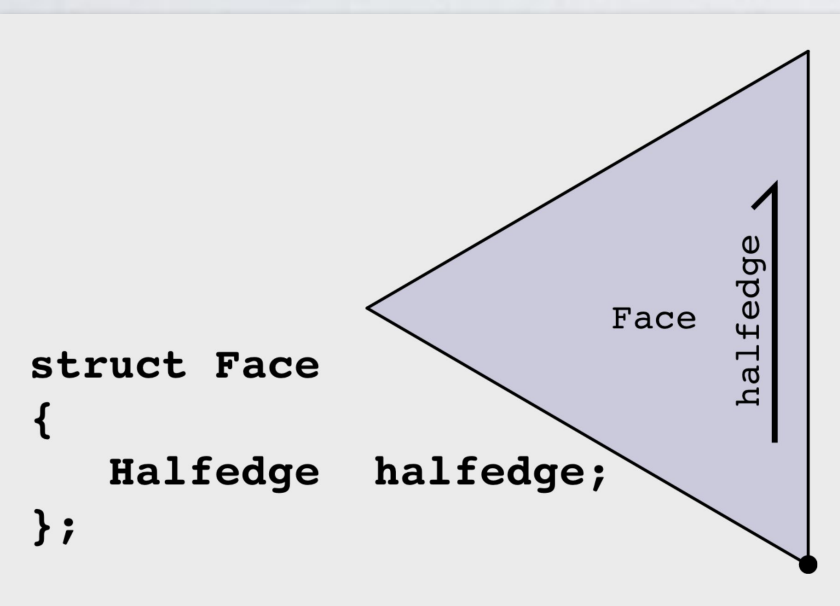
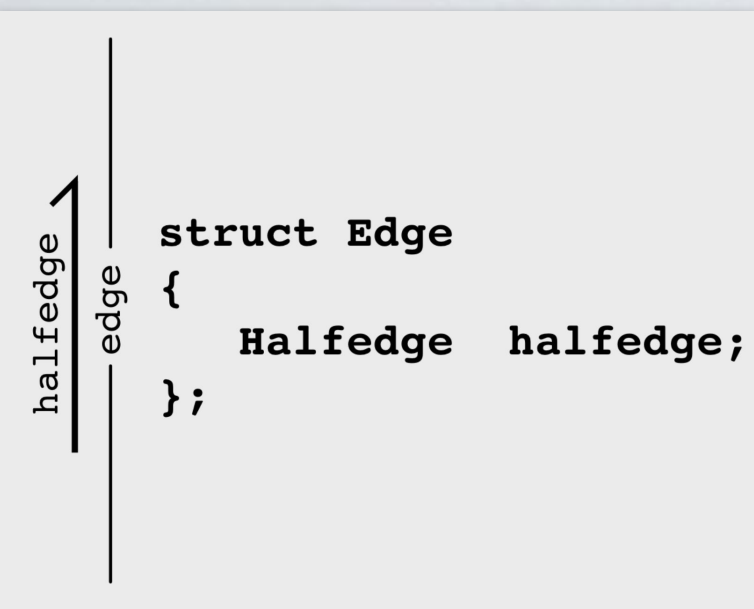
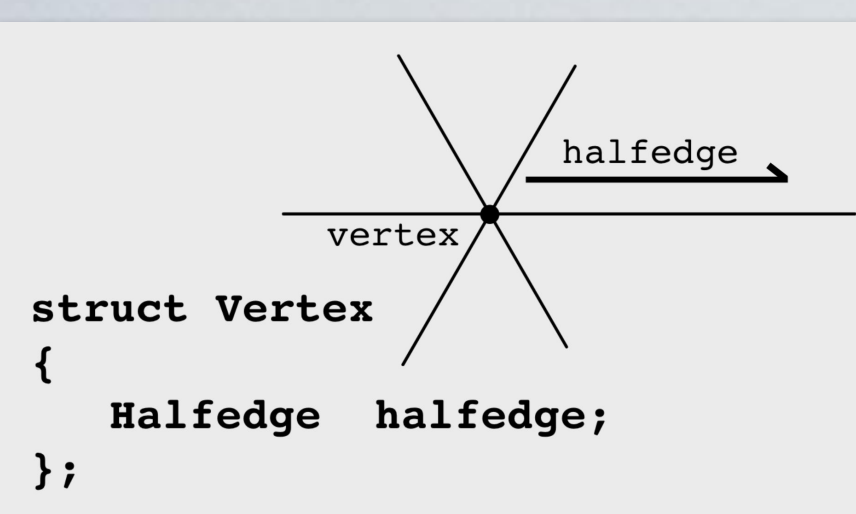
Face right_face = he.twin.face;

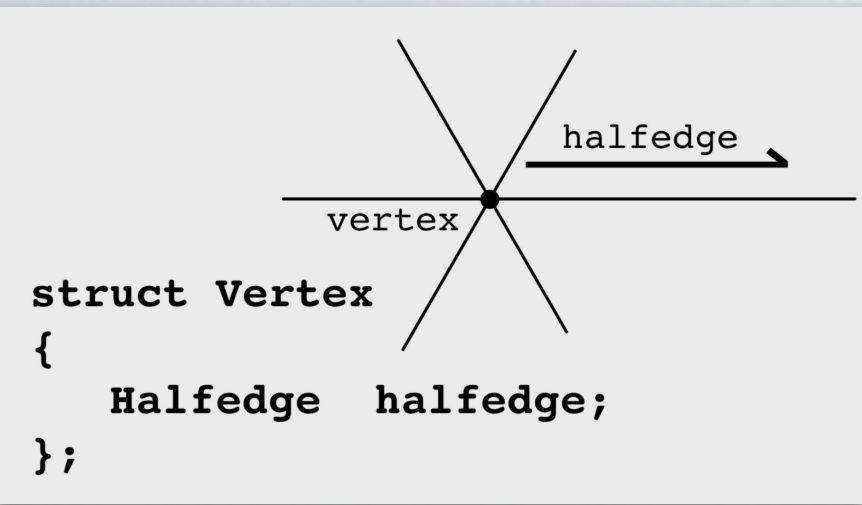


The Halfedge Data Structure

How would I find the edges adjacent to a triangle?

Given: `Face tri`

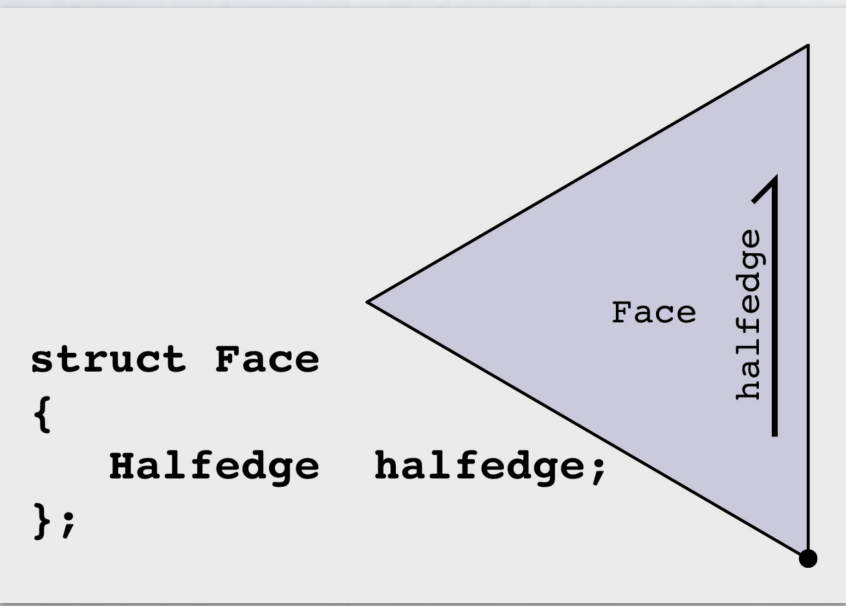
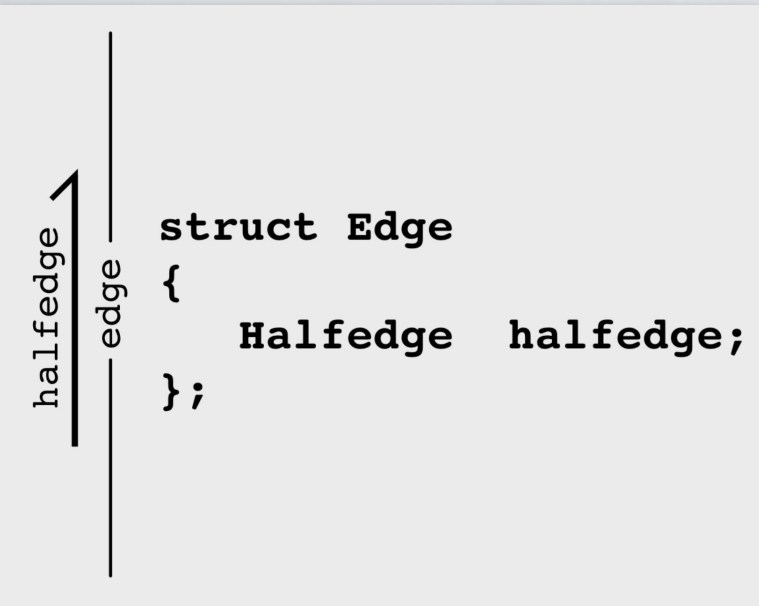




The Halfedge Data Structure

How would I find the edges adjacent to a triangle?

Given: `Face tri`

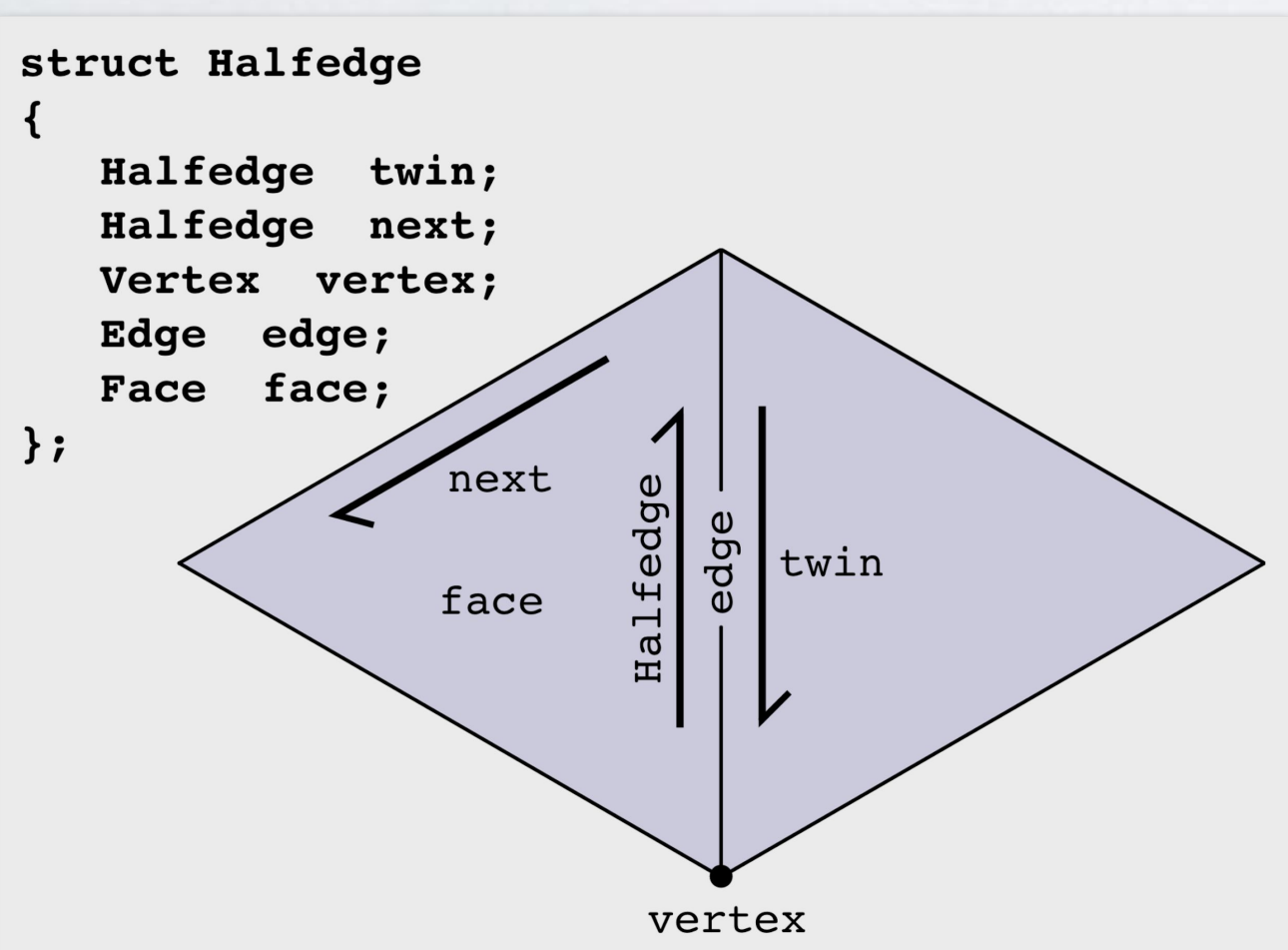


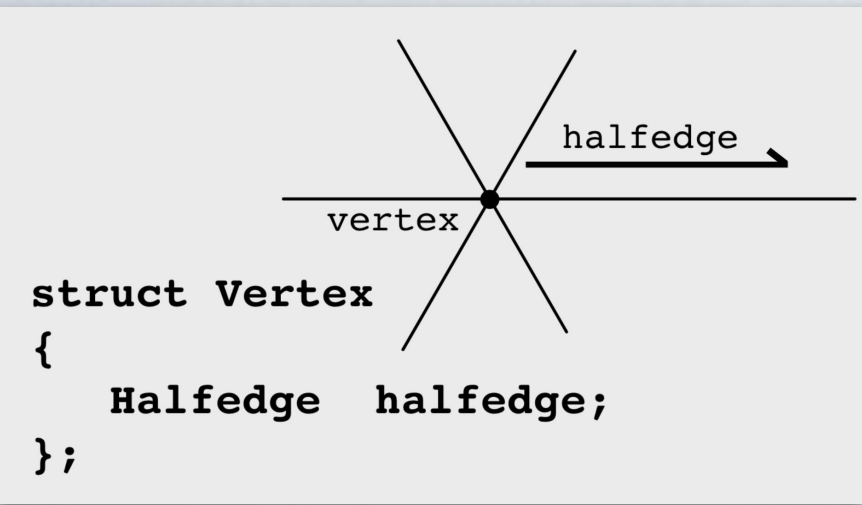
`Halfedge he = tri.halfedge;`

`Edge e1 = he.edge;`

`Edge e2 = he.next.edge;`

`Edge e3 = he.next.next.edge;`

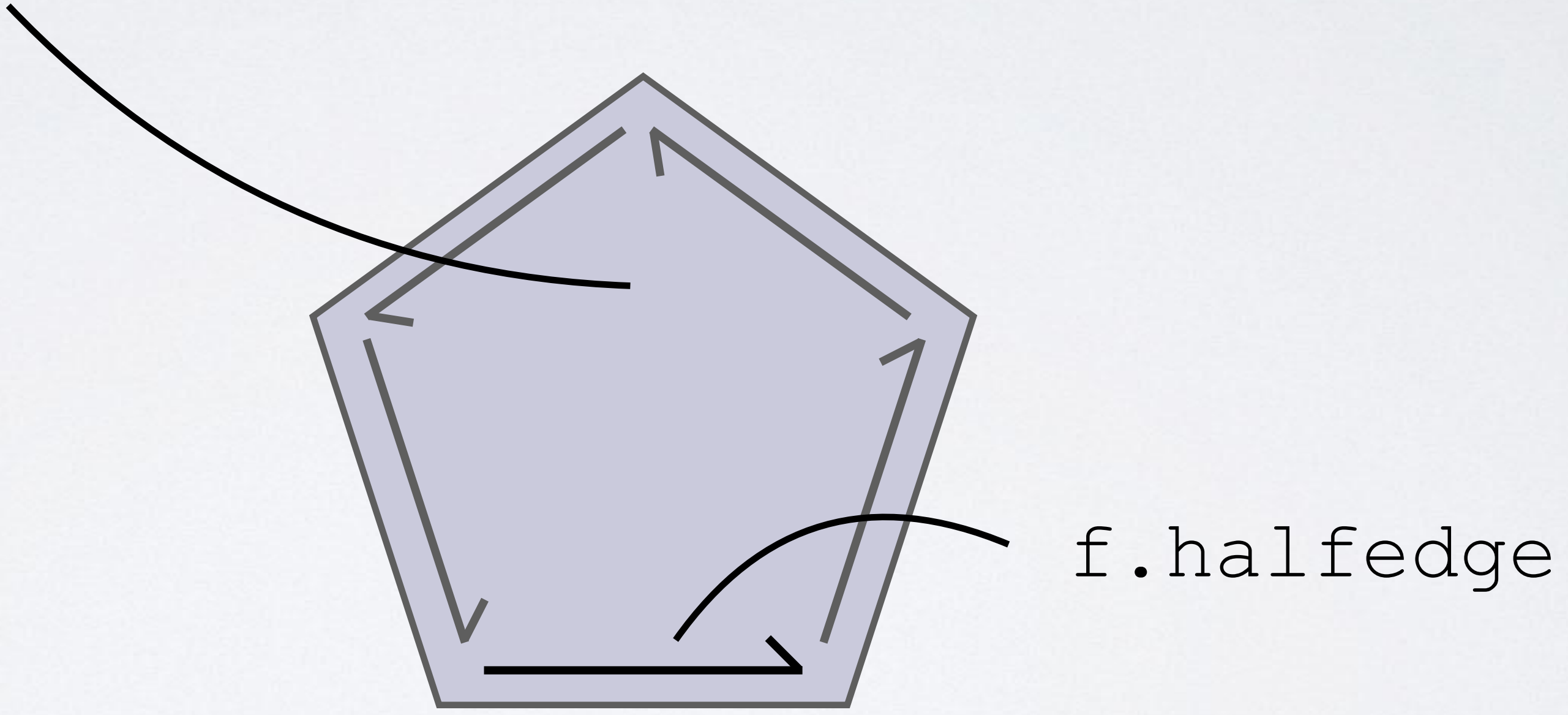
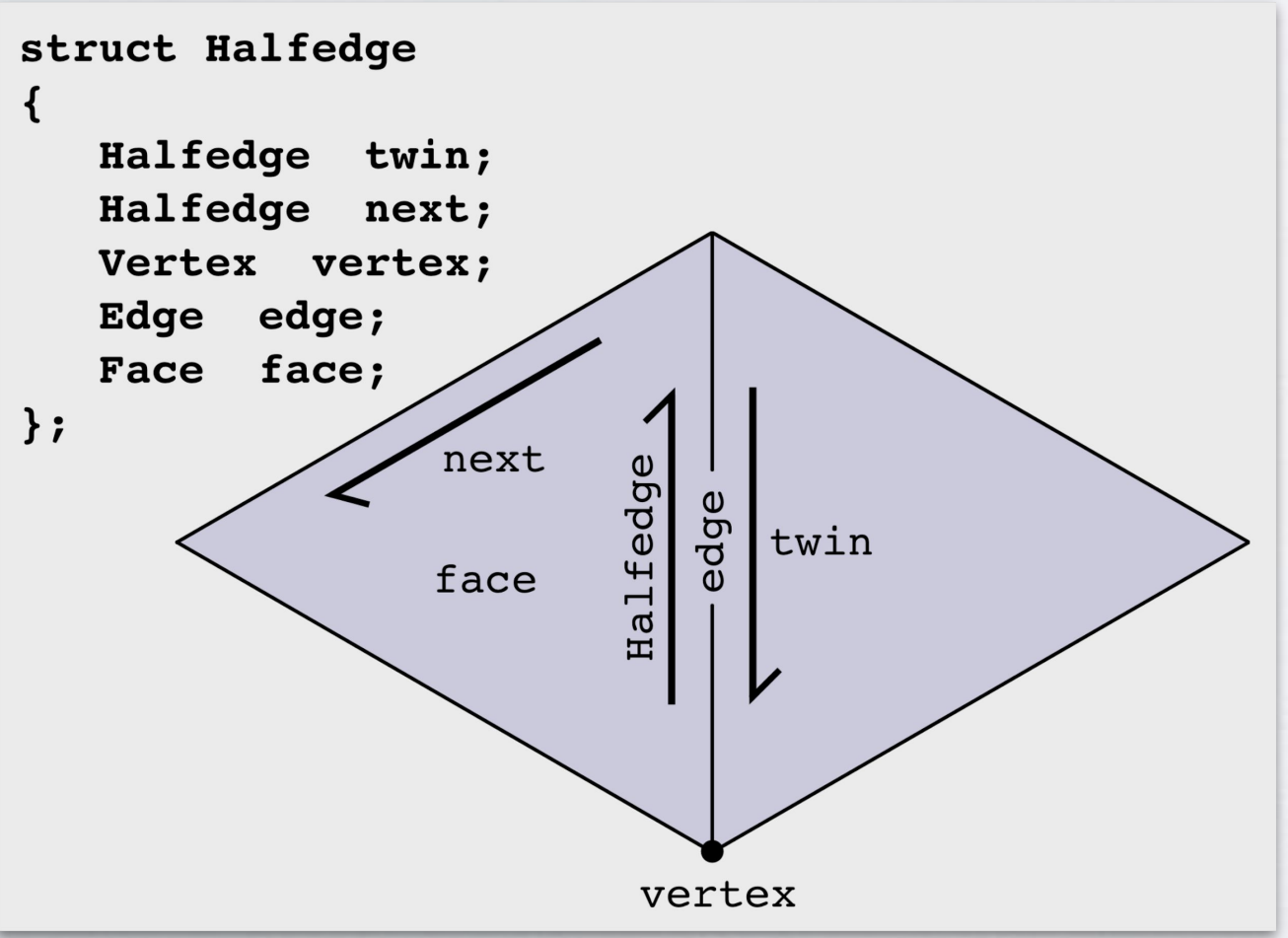
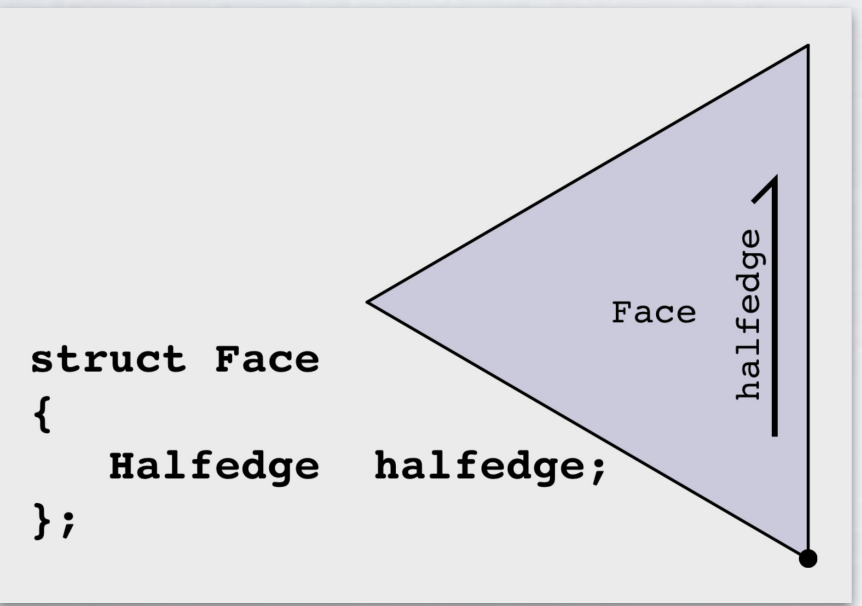
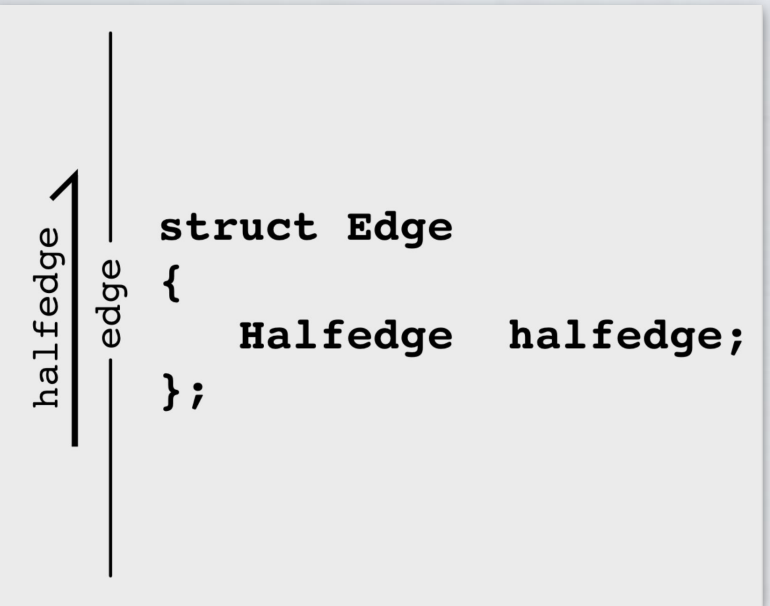


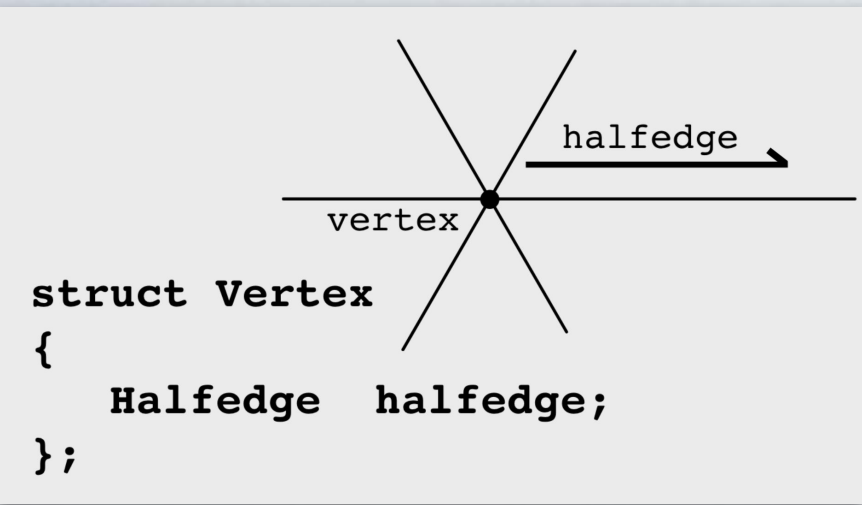


The Halfedge Data Structure

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

Given: **Face** *f*

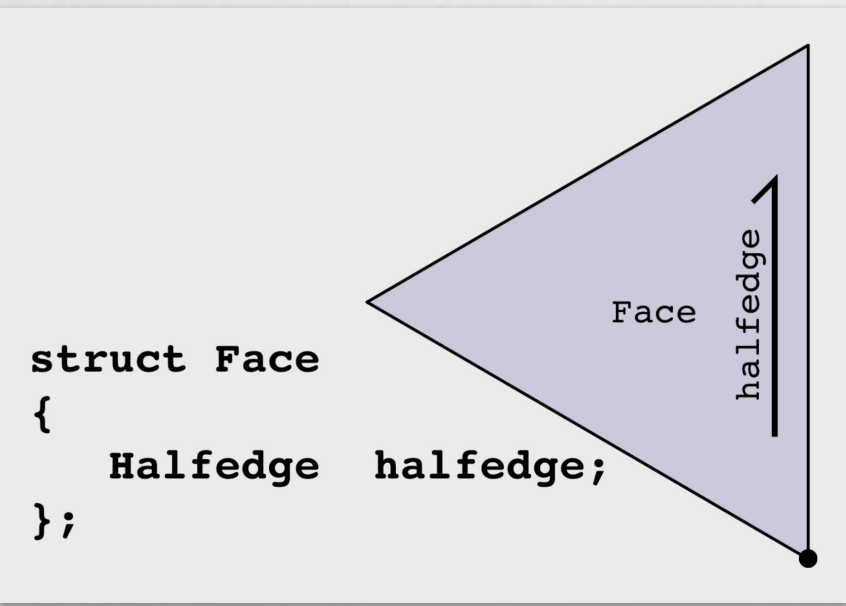
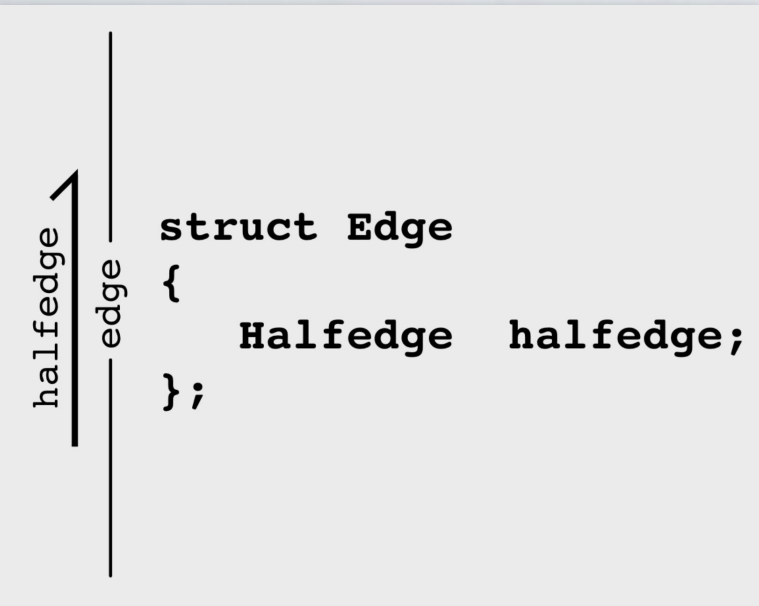




The Halfedge Data Structure

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

Given: `Face f`



```
Halfedge start = f.halfedge;
```

```
Halfedge he = start;
```

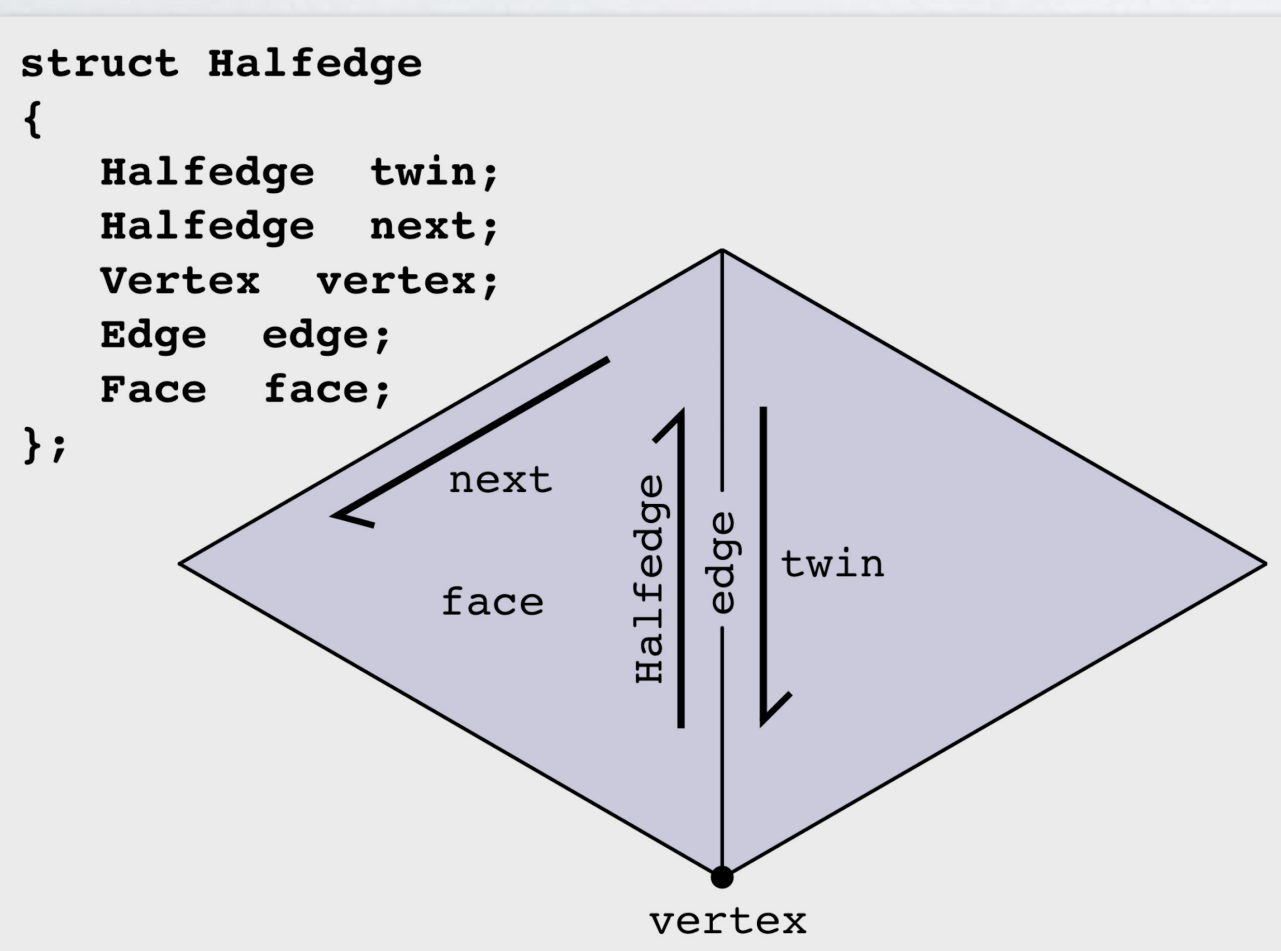
```
do {
```

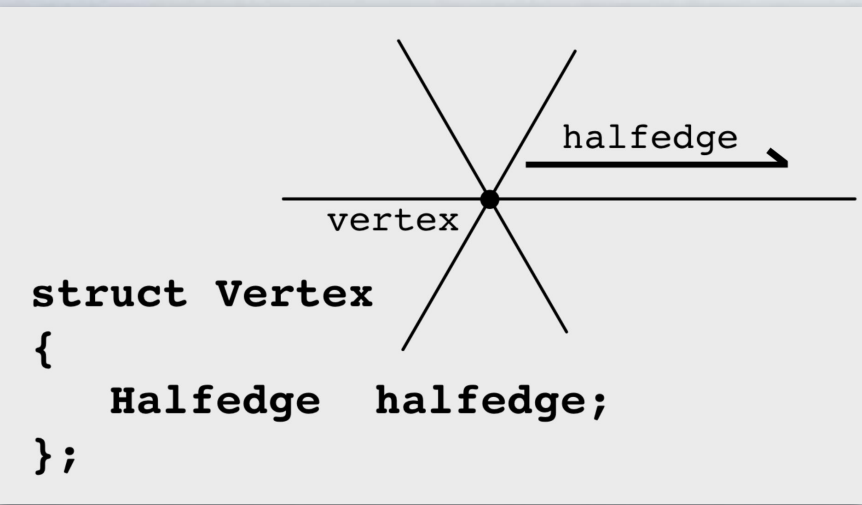
```
    Edge e = he.edge;
```

```
    /* Some code */
```

```
    he = he.next;
```

```
} while (he != start);
```

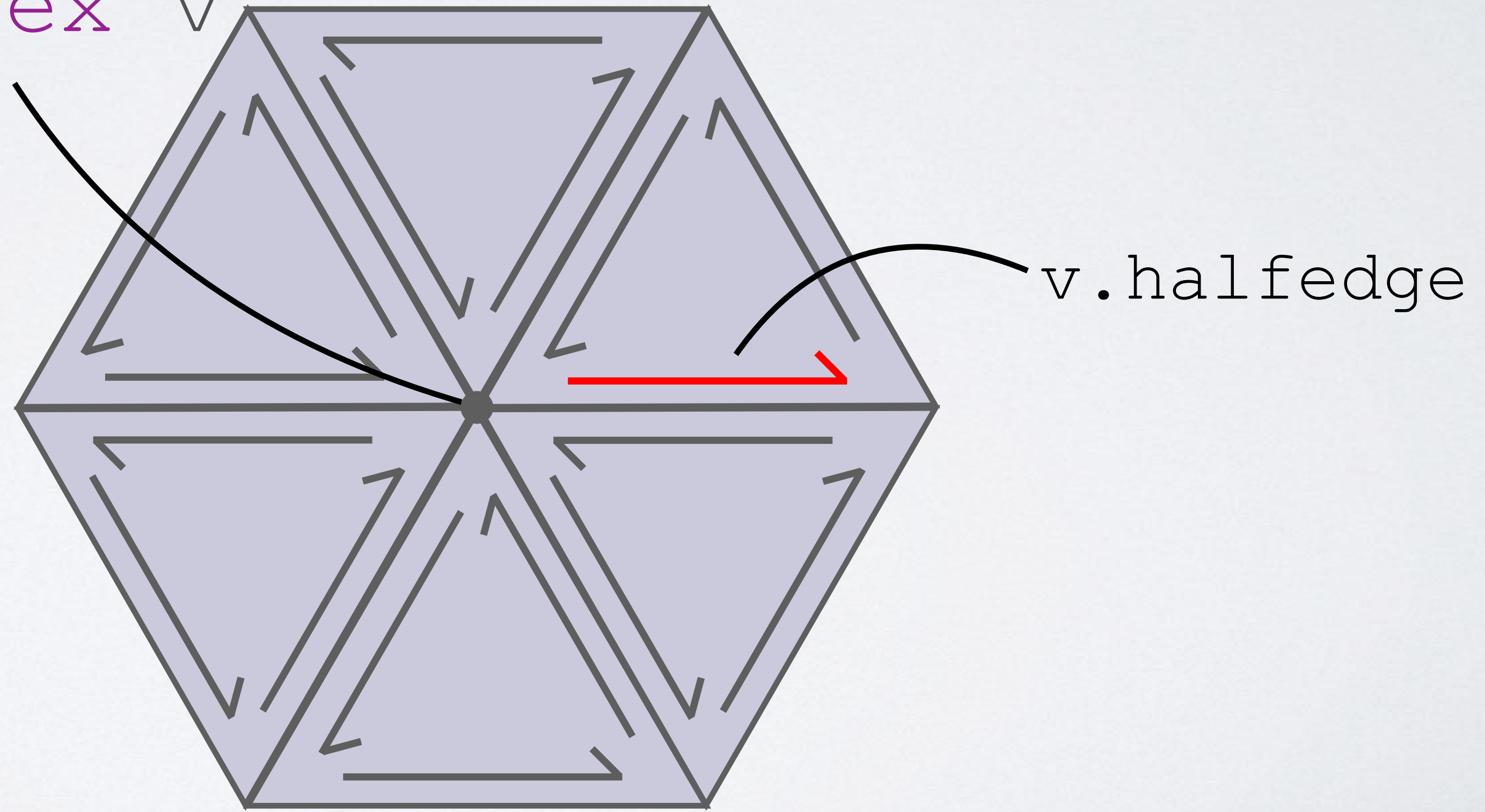
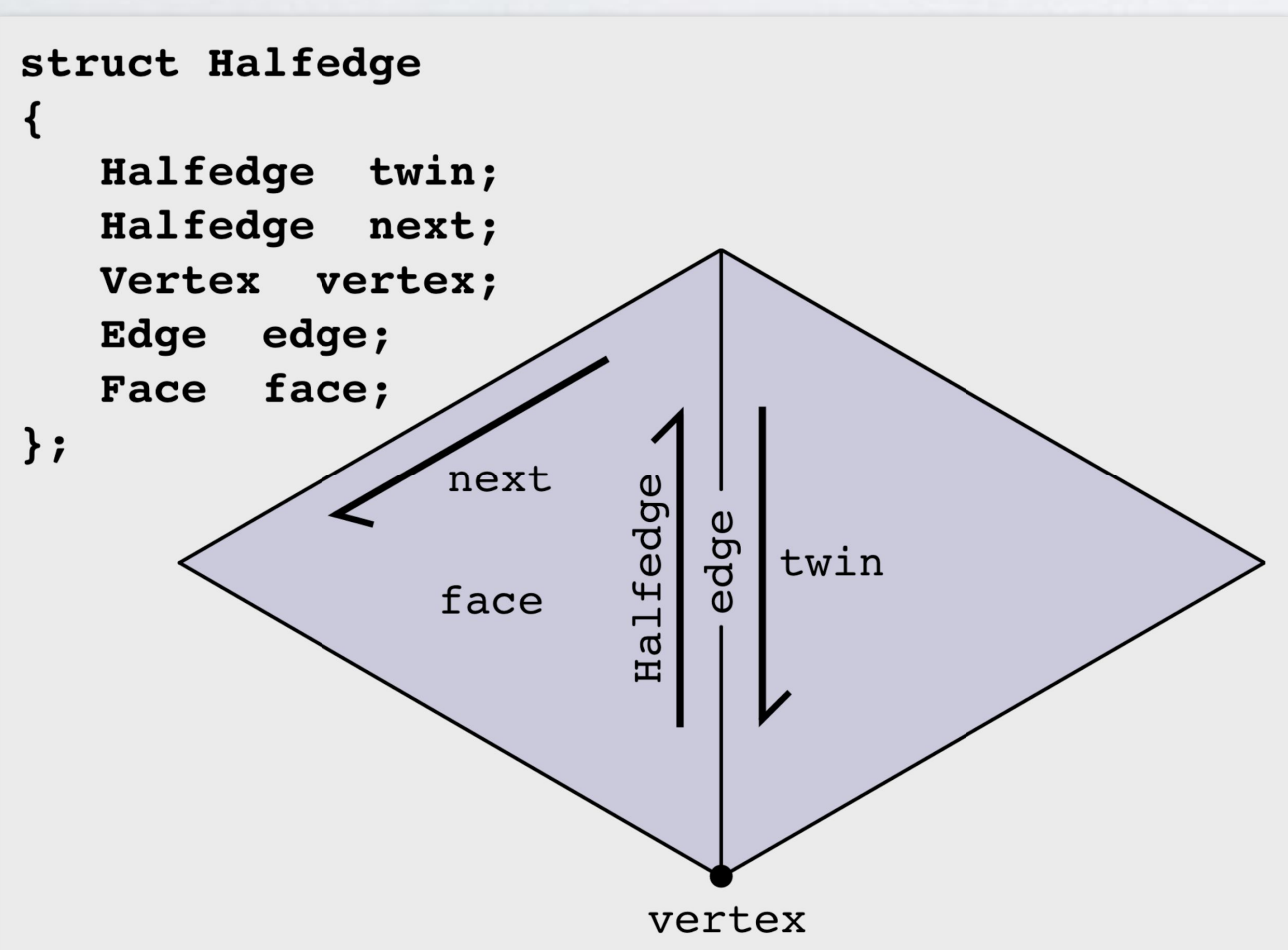
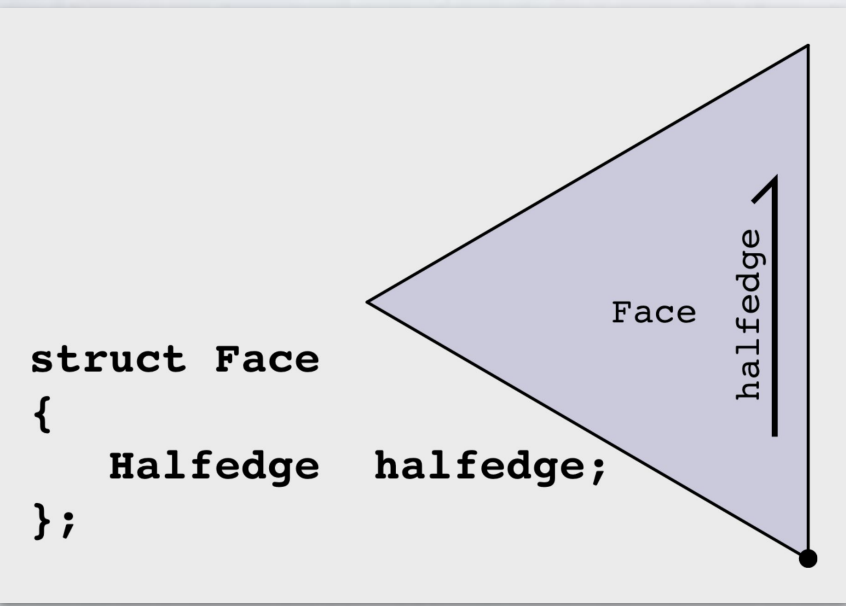
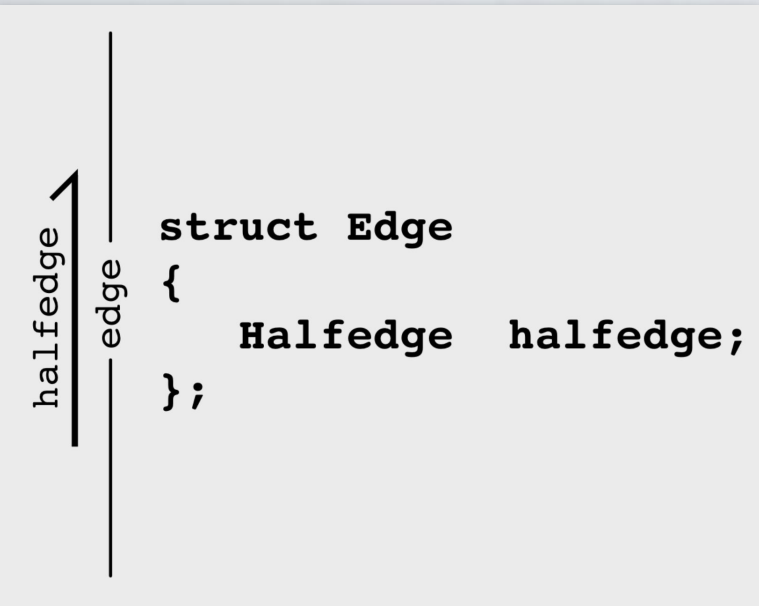


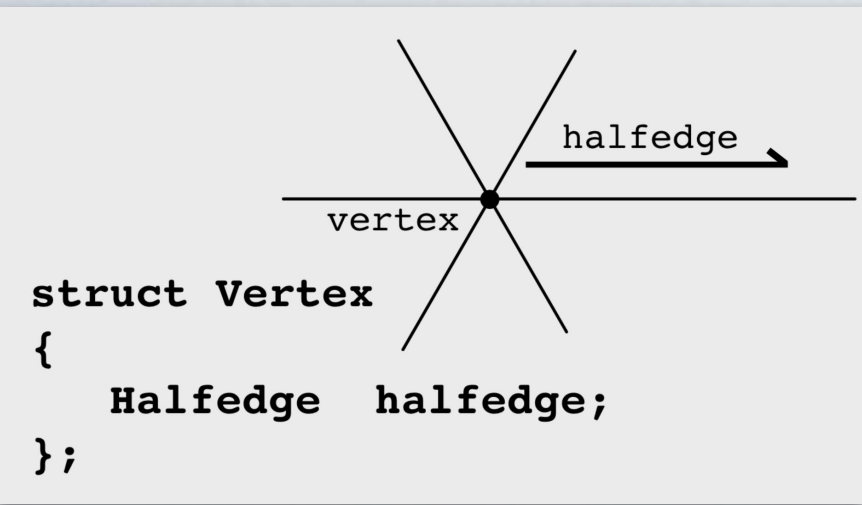


The Halfedge Data Structure

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;
```

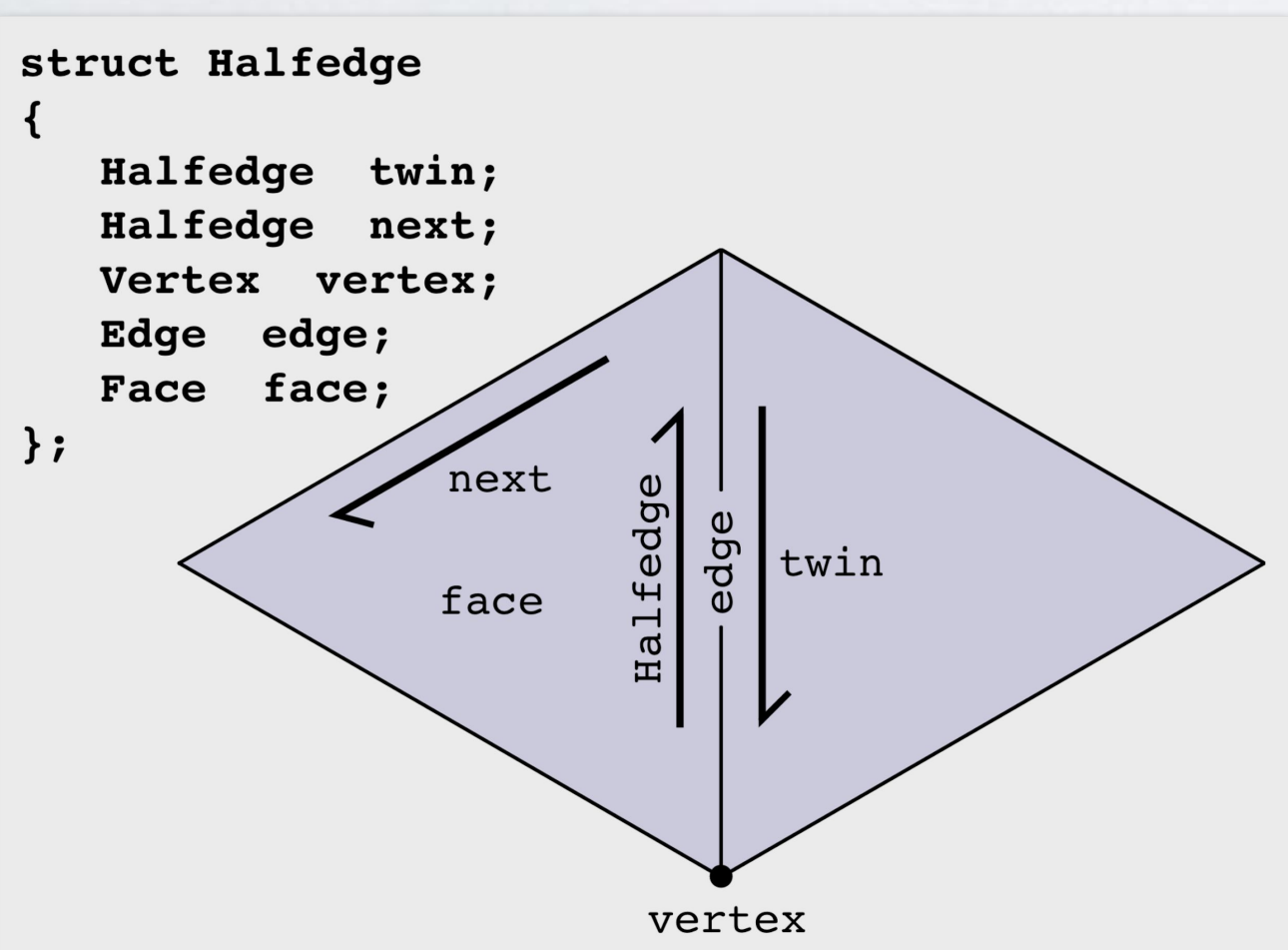
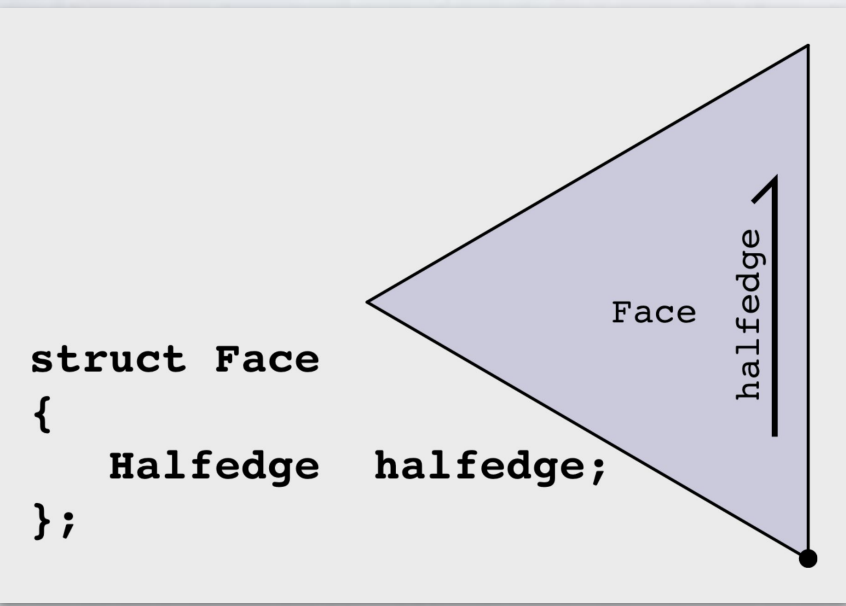
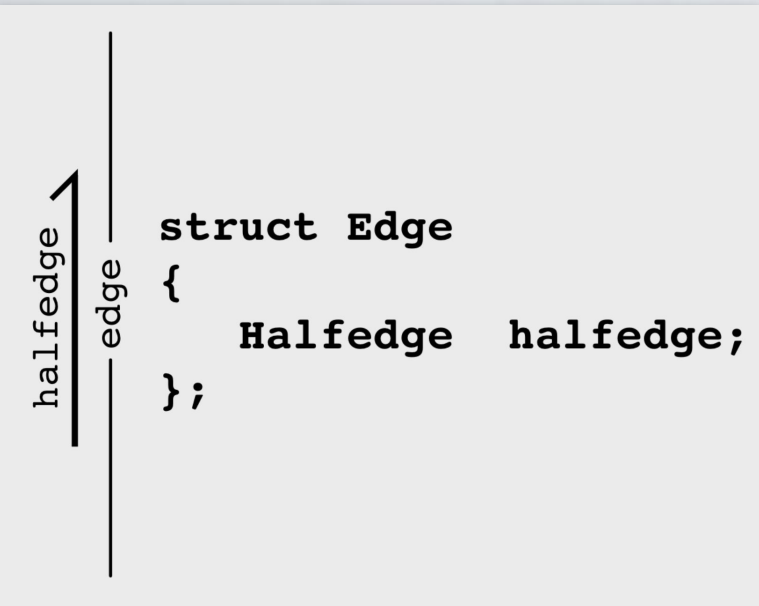
```
do {
```

```
  Edge e = he.edge;
```

```
  /* Some code */
```

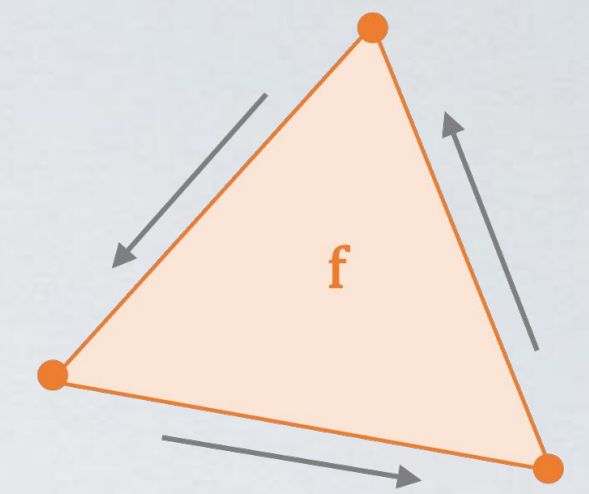
```
  he = he.twin.next;
```

```
} while (he != start);
```

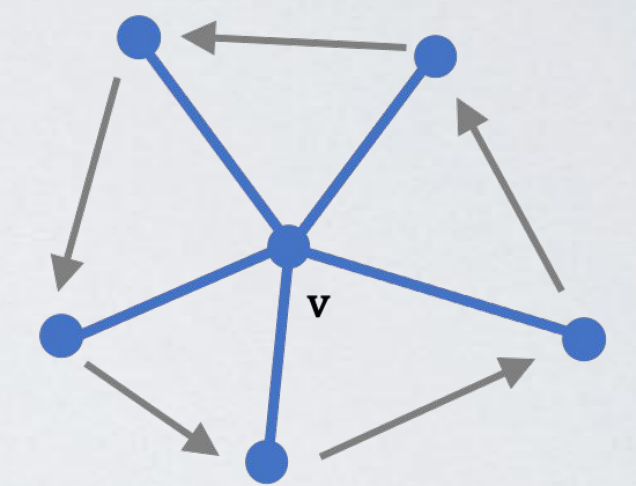


Many convenience functions in both JS and C++!

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

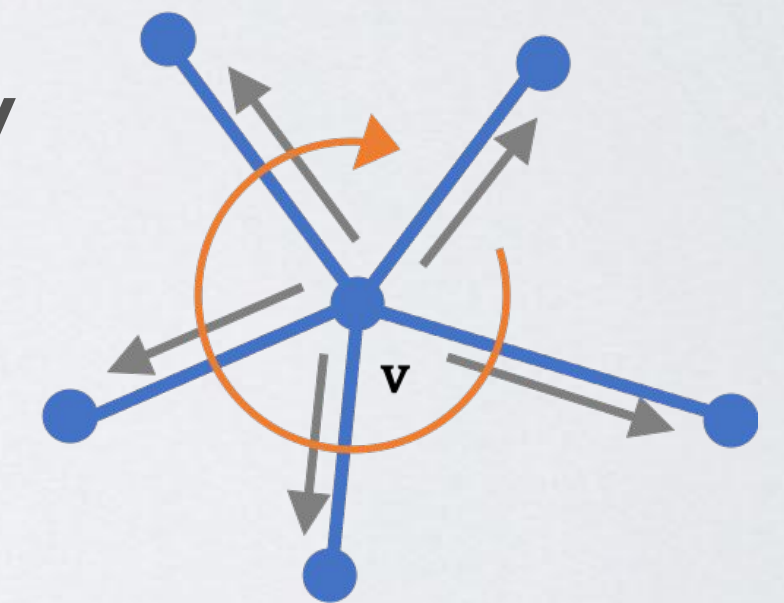


`v.adjacentVertices()` → iterator over vertices adjacent to vertex `v`



`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?

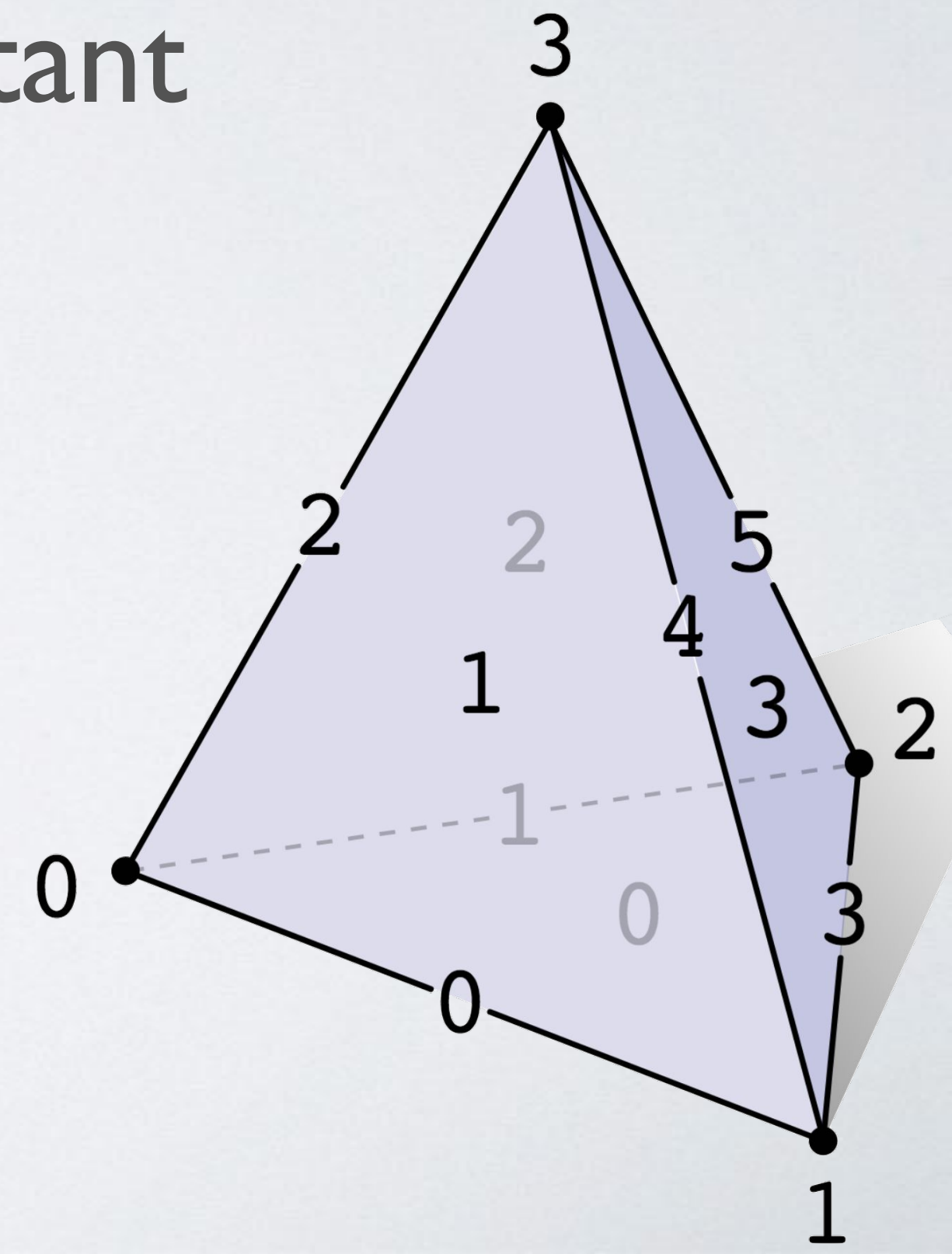
- Option 2: 2D array
- If your matrix doesn't have much structure, this might be the best you can do
- But it can take a lot of space to write down an entire matrix
- And working with (really) big matrices is slow

Matrices

- What matrices do we care about?
- It turns out that *adjacency matrices* are very important

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

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



Matrices

- Most entries are 0!
- We can improve our lives by only storing nonzero entries → sparse matrices

The screenshot shows the documentation for the `SparseMatrix` class in the `geometry-processing-js` library. The page title is "Class: SparseMatrix" and it is located under the namespace `LinearAlgebra.SparseMatrix`. The documentation includes a description: "This class represents a m by n matrix where only nonzero entries are stored explicitly. Do not use the constructor, instead use static factory methods." It also provides an "Example" section with a code snippet:

```
1 let T = new Triplet(1, 1, 3.4);
2 T.addEntry(3.4, 11, 1);
3 T.addEntry(6.4, 99, 1);
4 let A = SparseMatrix.fromTriplets(T);
5
6 let B = SparseMatrix.zeros(10, 10);
7
8 let d = DenseMatrix.fromArray([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);
9 let C = SparseMatrix.multiply(A, B);
```

Below the example, there is a "Methods" section. A callout box highlights the `SparseMatrix<T>` typedef, describing it as "A templated sparse matrix typedef, to Eigen's sparse matrix type." and providing the following code:

```
template <typename T>
using SparseMatrix = Eigen::SparseMatrix<T>;
```

It also notes: "Use like `SparseMatrix<double>` OR `SparseMatrix<int>`."

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 \Rightarrow 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

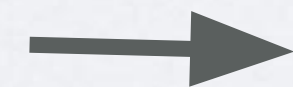


Solving Linear Systems

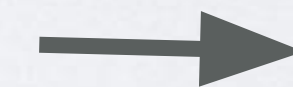
Linear Systems of Equations

Linear algebra review

$$\begin{array}{rcl} x + 2y - 4z & = & 1 \\ 3x - 5y + 7z & = & 2 \\ -x + 3y + 5z & = & -2 \end{array}$$



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



$$Ax = b$$

A x 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?

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

$$\begin{aligned} x &= 1 \\ x + 2y &= 5 \implies y = 2 \\ 2x + 3y - 3z &= 11 \implies z = -1 \end{aligned}$$

- (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

Search or jump to... Pull requests Issues Marketplace Explore

cmu-geometry / ddg-exercises-js Watch 1 Star 15 Fork 6

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

No description, website, or topics provided. Edit

Manage topics

4 commits 1 branch 0 releases 1 contributor MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

Mark Gillespie and Mark Gillespie Fixed function names to match assignment Latest commit 9db07fb 6 days ago

core	Added simplicial complex assignment	6 days ago
docs	Fixed function names to match assignment	6 days ago
imgs	first commit	a year ago
input	Added simplicial complex assignment	6 days ago
libs	first commit	a year ago
linear-algebra	first commit	a year ago
projects	Fixed function names to match assignment	6 days ago
style	first commit	a year ago
tests	Fixed function names to match assignment	6 days ago
utils	first commit	a year ago
LICENSE	first commit	a year ago
README.md	first commit	a year ago

README.md

- 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

geometry-processing-js

Modules ▾

Classes ▾

Global ▾

Search



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

```
1 git clone https://github.com/cmu-geometry/ddg-exercises-js.git
2 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 included

`ddg-exercises-js/docs/index.html`

- Coding assignments

`ddg-exercises-js/projects`

- Tests

`ddg-exercises-js/tests`

Documentation

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
edges	Array.<module:Core.Edge>	The edges
faces	Array.<module:Core.Face>	The faces
corners	Array.<module:Core.Corner>	The corners
halfedges	Array.<module:Core.Halfedge>	The halfedges
boundaries	Array.<module:Core.Face>	The boundaries of the mesh.
generators	Array.<Array.<module:Core.Halfedge>>	An array of generators representing the 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

```
1 let T = new Triplet(100, 100);
2 T.addEntry(3.4, 11, 43);
3 T.addEntry(6.4, 99, 99);
4 let A = SparseMatrix.fromTriplet(T);
5
6 let B = SparseMatrix.identity(10, 10);
7
8 let d = DenseMatrix.ones(100, 1);
9 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 Search

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

```
1 git clone https://github.com/cmu-geometry/ddg-exercises-js.git
2 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).

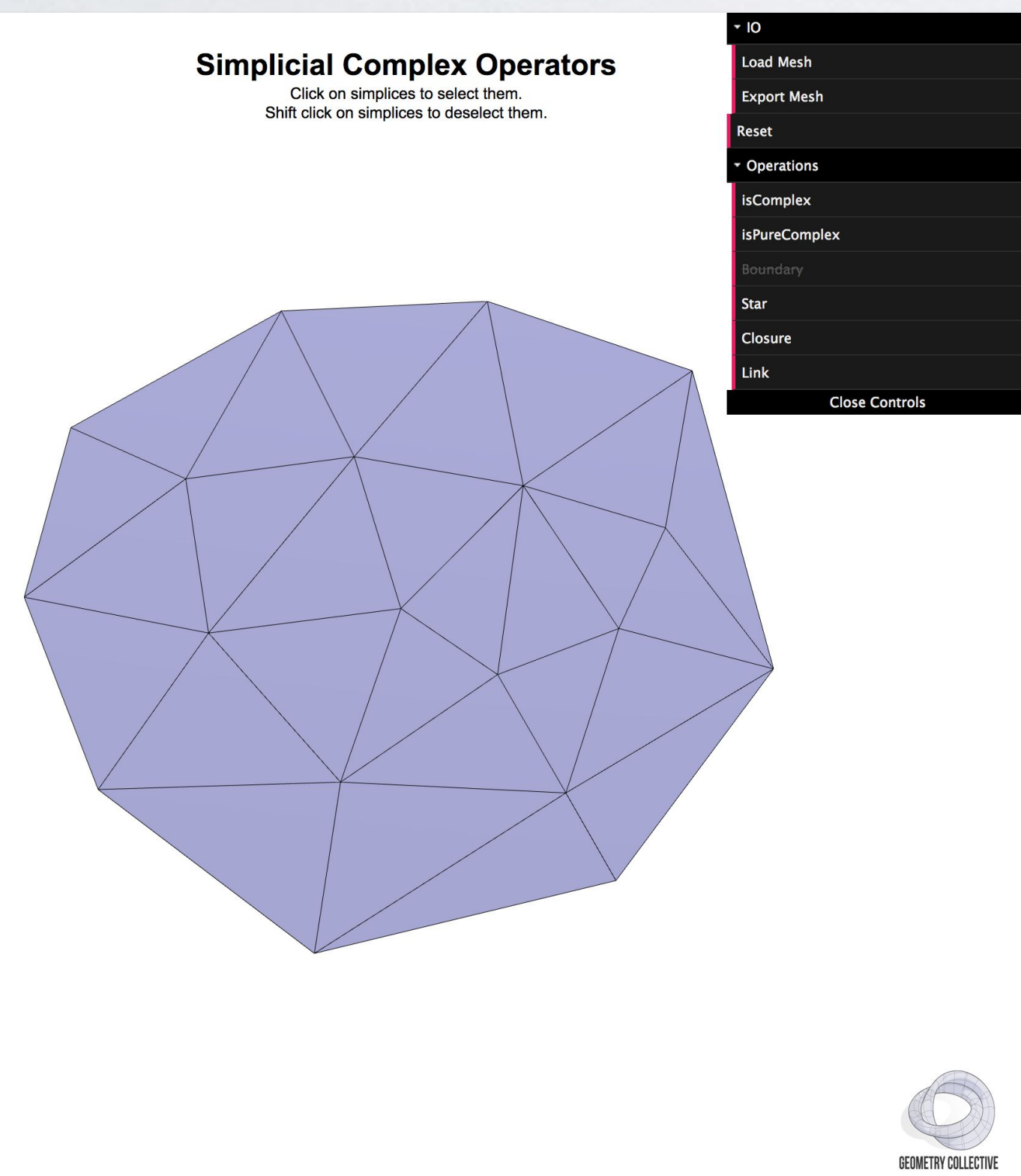
Dependencies (all included)

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`

passes: 5 failures: 41 duration: 0.09s 100%

passes: 46 failures: 0 duration: 0.53s 100%

Simplicial Complex Operators

isComplex

- ✓ A vertex 69ms
- ✓ An edge
- ✓ A closed edge
- ✓ A face
- ✓ A face with its edges
- ✓ A closed face

pureDegree

- ✓ A vertex
- ✓ An edge
- ✓ A closed edge
- ✓ A face
- ✓ A face with its edges
- ✓ A closed face
- ✓ A closed face and closed edge

A0

- ✓ Has $|E|$ rows
- ✓ Has $|V|$ columns
- ✓ Rows sum to two
- ✓ Columns sum to degrees

A1

- ✓ Has $|F|$ rows
- ✓ Has $|E|$ columns
- ✓ Rows sum to number of faces
- ✓ Columns sum to two

buildVertexVector

- 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 <u>homology generators</u> .

Methods

geometry-processing-js Modules Cla

Class: Halfedge

Core. Halfedge

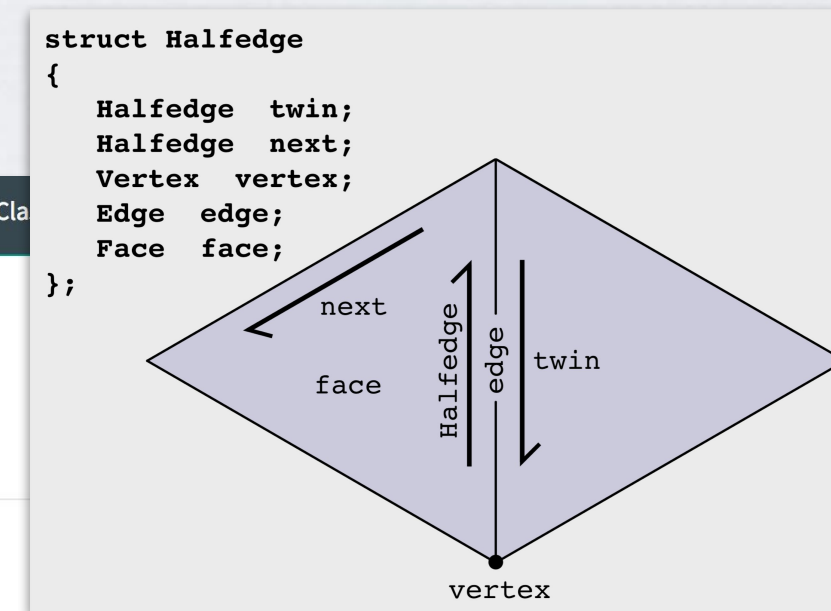
new Halfedge()

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

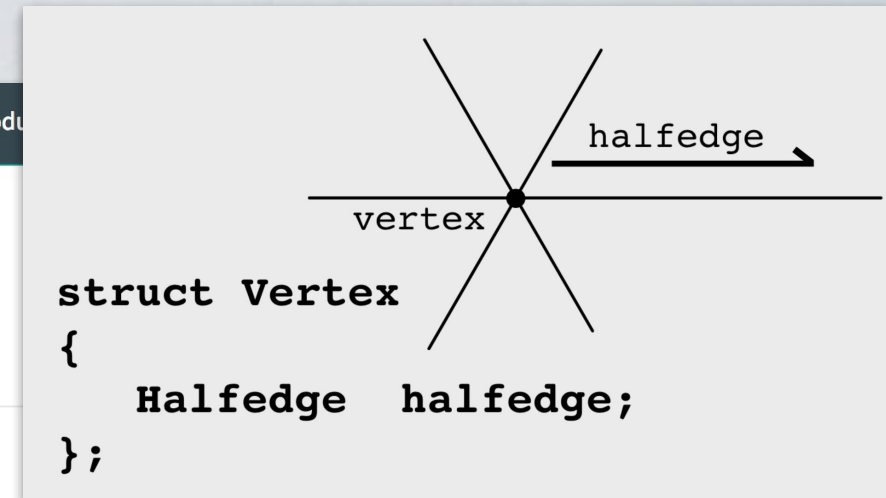
Core. Vertex

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

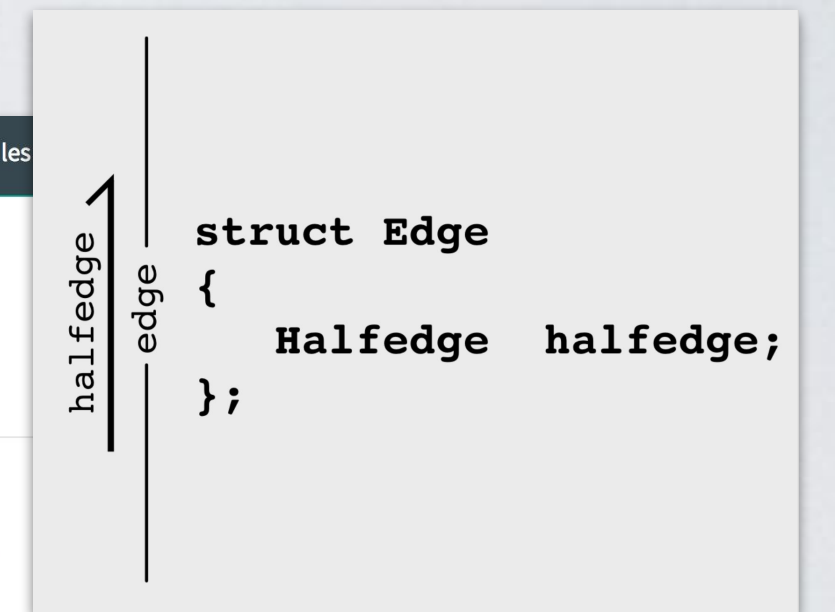
Core. Edge

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

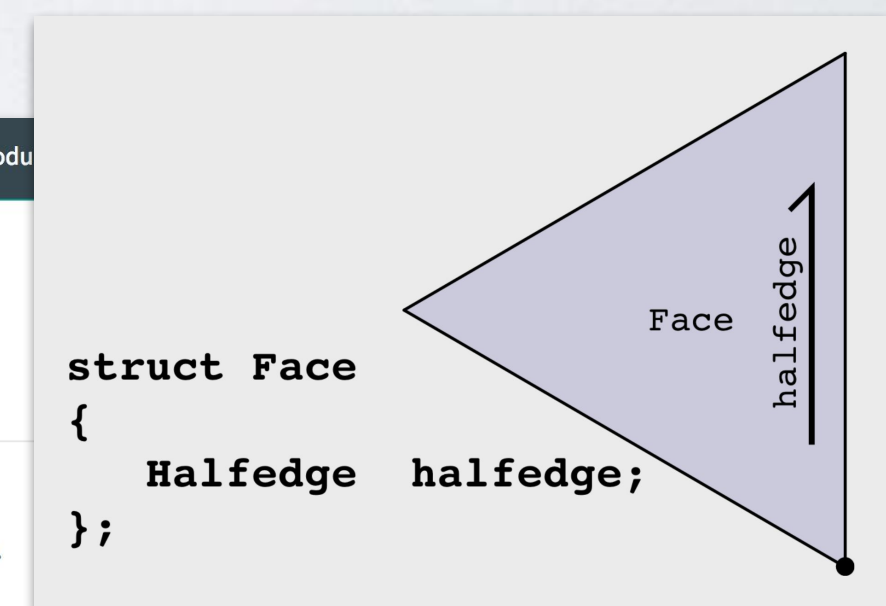
Core. Face

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

```
1 let f = mesh.faces[0]; // or let b = mesh.boundaries[0]
2 for (let v of f.adjacentVertices()) {
3   // Do something with v
4 }
```

adjacentEdges()

Convenience function to iterate over the edges adjacent to this vertex.

Returns:

Type
module:Core.Edge

Example

```
1 let v = mesh.vertices[0];
2 for (let e of v.adjacentEdges()) {
3   // Do something with e
4 }
```

— raw 5



Linear algebra in `ddg-exercises-js`

Sparse Matrices in ddg-exercises-js

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

```
1 let T = new Triplet(100, 100);
2 T.addEntry(3.4, 11, 43);
3 T.addEntry(6.4, 99, 99);
4 let A = SparseMatrix.fromTriplet(T);
5
6 let B = SparseMatrix.identity(10, 10);
7
8 let d = DenseMatrix.ones(100, 1);
9 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.

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

geometry-processing-js Modules Classes Global Search

Class: Triplet

LinearAlgebra. Triplet

new Triplet(m, n)

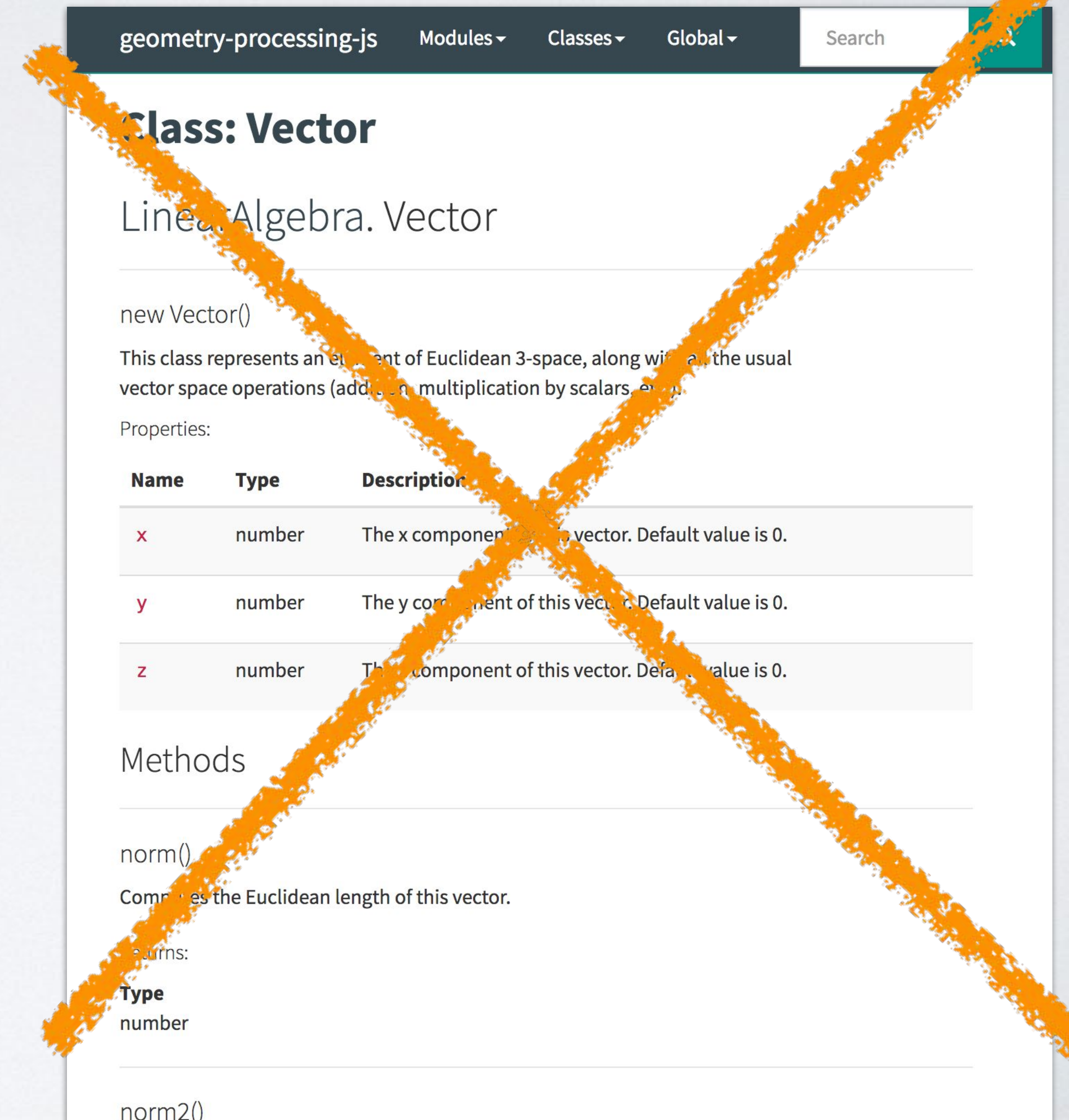
This class represents a small structure to hold nonzero entries in a SparseMatrix. Each entry is a triplet of a value and the (i, j)th indices, i.e., (x, i, j).

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`



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

Class: Vector

LinearAlgebra.Vector

`new Vector()`

This class represents an element of Euclidean 3-space, along with all the usual vector space operations (addition, multiplication by scalars, etc.).

Properties:

Name	Type	Description
<code>x</code>	number	The x component of this vector. Default value is 0.
<code>y</code>	number	The y component of this vector. Default value is 0.
<code>z</code>	number	The z component of this vector. Default value is 0.

Methods

`norm()`

Computes the Euclidean length of this vector.

Returns:

Type
number

`norm2()`

Solving linear systems

Cholesky

LU

QR

geometry-processing-js Modules Classes Global Search

Class: Cholesky

LinearAlgebra. Cholesky

new Cholesky()

This class represents a Cholesky LL^T factorization of a square SparseMatrix. The factorization is computed on the first call to solvePositiveDefinite (e.g. when only the right hand side b of the linear system $Ax = b$ changes) unless the

chol()

Returns a sparse Cholesky factorization of this sparse matrix.

Returns:

Type

module:LinearAlgebra.Cholesky

```
5 let llt = A.chol()
6 let x = llt.solve(b)
7
8 b.scaleBy(5);
9 x = llt.solveP
```

Methods

solvePositiveDefi

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

Parameters:

Name	Type	Description
b	module:LinearAlgebra.DenseMatrix	The dense right hand side b .

geometry-processing-js Modules Classes Global Search

Class: LU

LinearAlgebra. LU

new LU()

This class represents a LU factorization of a square SparseMatrix. The factorization is computed on the first call to 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

lu()

Returns a sparse LU factorization of this sparse matrix.

Returns:

Type

module:LinearAlgebra.LU

qr()

Returns a sparse QR factorization of this sparse matrix.

Returns:

Type

[module:LinearAlgebra.QR](#)

geometry-processing-js Modules Classes Global Search

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

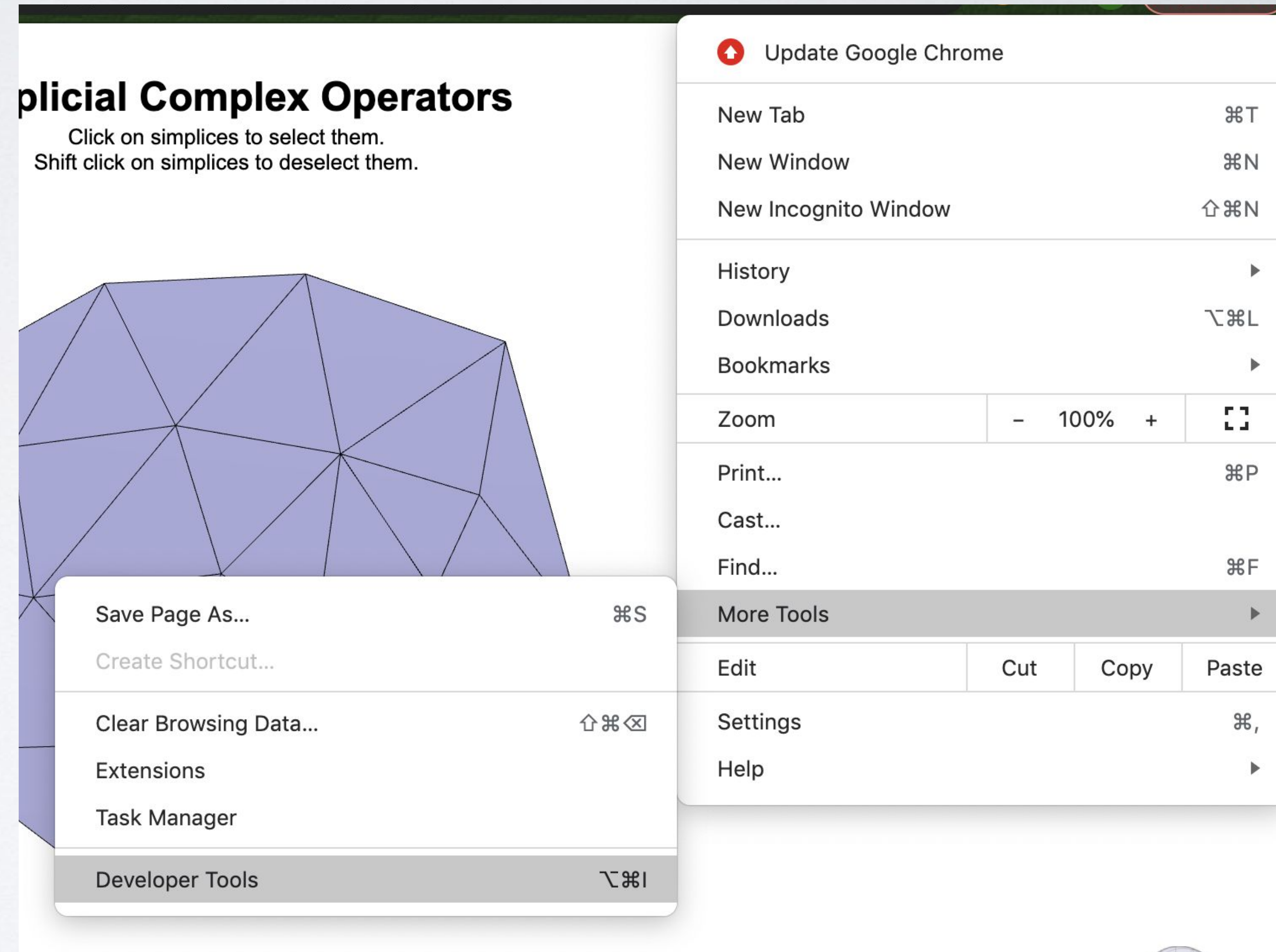
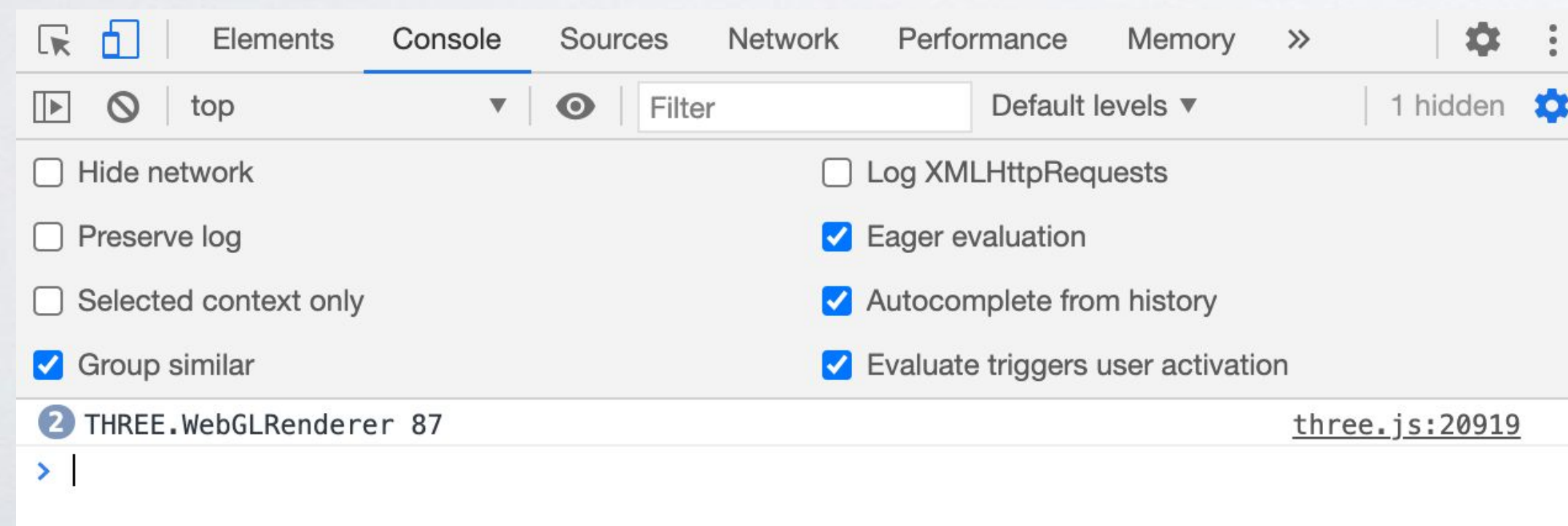
```
- raw
= b, where A is a rectangular sparse matrix
(5, 5);
);
```

tion is reused

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++)

The image shows two screenshots of GitHub project pages. The top screenshot is for the Geometry Central repository, which is a modern C++ library for geometry processing. The bottom screenshot is for the Polyscope repository, a C++/Python viewer for 3D data. Both pages feature navigation menus, search bars, and lists of features or documentation links.

Geometry Central

Home

- Build
- Tutorials
- Surface
- Point Cloud
- Numerical
- Utilities

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.

linux passing macOS passing windows passing

Features include:

- A polished **surface mesh** class, with efficient support for mesh 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 - C++

Home

- About
- Building
- Basics
- Data Adaptors
- Features
- Structures
- Integrations

POLYSCOPE

license MIT stars 580 pypi v0.1.6 linux passing macOS passing windows passing

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

GeometryCollective / **ddg-exercises**

<> Code Issues Pull requests Actions Projects Wiki Security Insights

main 1 branch 0 tags Go to file Add file Code

nzfeng Update README 919c88f 14 days ago 11 commits

core	Fix bug in eulerCharacteristic()	15 days ago
deps	Fix bug in eulerCharacteristic()	15 days ago
imgs	First commit of assignments and submodules	15 days ago
input	First commit of assignments and submodules	15 days ago
projects	Update README and resize some vectors; add submodules	15 days ago
utils	First commit of assignments and submodules	15 days ago
.clang-format	First commit of assignments and submodules	15 days ago
.gitignore	First commit of assignments and submodules	15 days ago
.gitmodules	First commit of assignments and submodules	15 days ago
README.md	Update README	14 days ago

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

```
CMake Error at CMakeLists.txt:72 (add_subdirectory):  
The source directory  
  
/Users/nicole/Downloads/ddg-exercises/deps/geometry-central  
does not contain a CMakeLists.txt file.
```

README.md

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.

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

Geometry Central

Search

Geometry Central

Home

Build

Tutorials

Surface

Point Cloud

Numerical

Utilities

Welcome to Geometry Central

Geometry-central is a modern C++ library of geometry processing, with a particular focus on surface mesh processing.

linux passing macOS passing windows passing

Features include:

- A polished **surface mesh** class, with efficient system of containers for associating data with mesh elements.
- Implementations of canonical **geometric quantities** (normals and curvatures to tangent vector bases) and differential geometry.
- A suite of **powerful algorithms**, including computing direction fields, and manipulating implicit surfaces.
- A coherent set of sparse **linear algebra tools**, based on Eigen, which automatically utilize better solvers if available on the system.

Geometry Central

- Home
- Build
- Tutorials
- Surface
 - Surface Mesh
 - Basics
 - Elements
 - Boundaries
 - Navigation and Iteration
- Containers
- Mutation
- Delta Complexes
- Internals
- Geometry
 - Overview
 - Quantities
- Utilities
- Algorithms
- Point Cloud
- Numerical
 - Matrix Types
 - Linear Algebra Utilities
 - Linear Solvers
- Utilities

- Detailed documentation at [https://geometry-central.net/!](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/`)

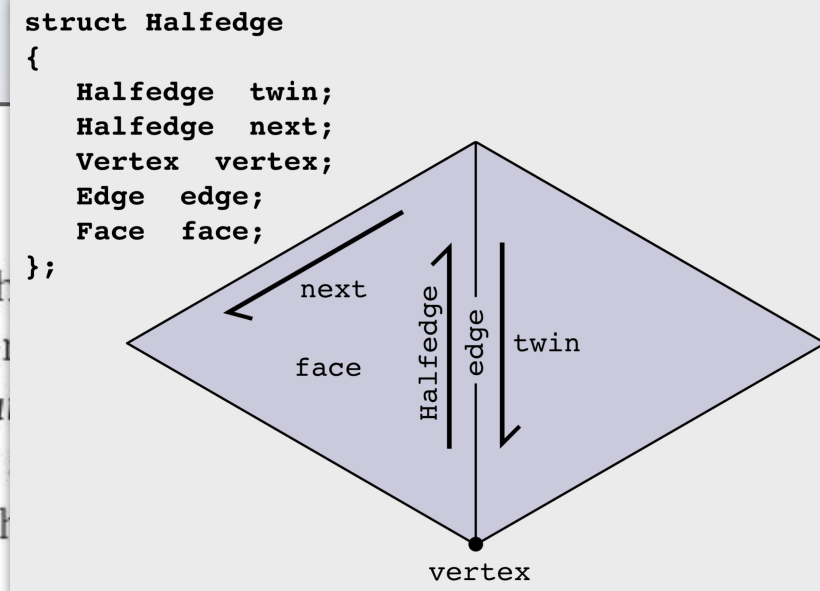


Navigating halfedges

In Geometry Central

Halfedge

A halfedge is the basic building block of a mesh. A halfedge is *half* of an *edge*, connecting two vertices in some face. The halfedge is directed, from its tail vertex to its tip vertex, in the clockwise orientation: the halfedges with in a face point in clockwise direction. On a `ManifoldSurfaceMesh`, a halfedge (and its twin 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()` >

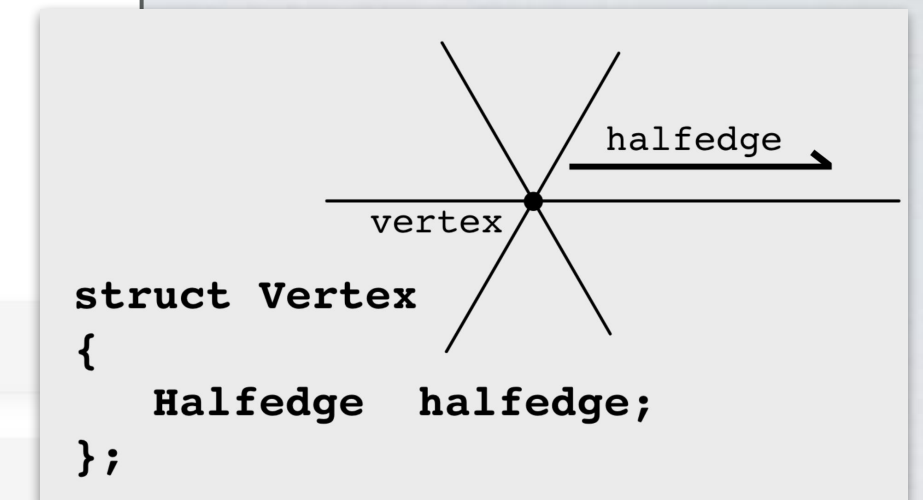
Vertex

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.

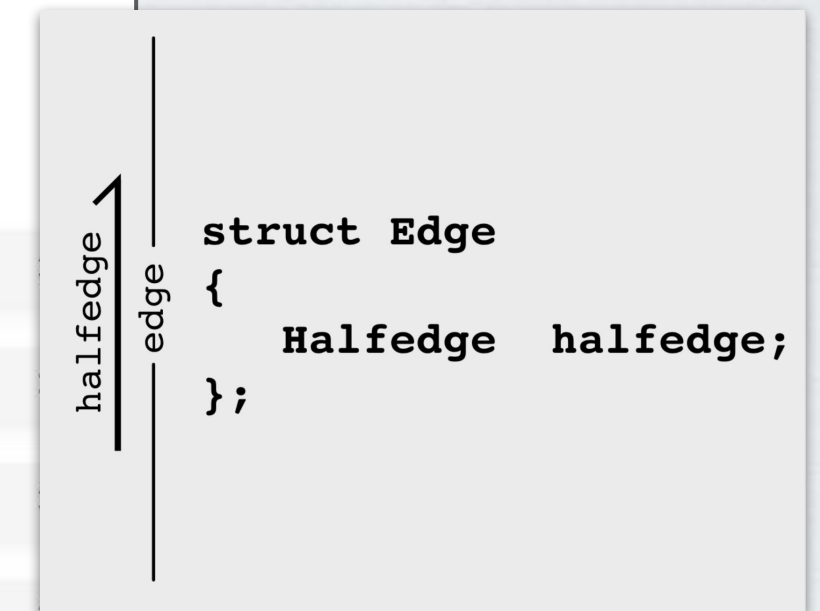
Traversal:

`Halfedge Edge::halfedge()`

`Vertex Edge::otherVertex(Vertex v)`

`Vertex Edge::firstVertex()`

`Vertex Edge::secondVertex()`



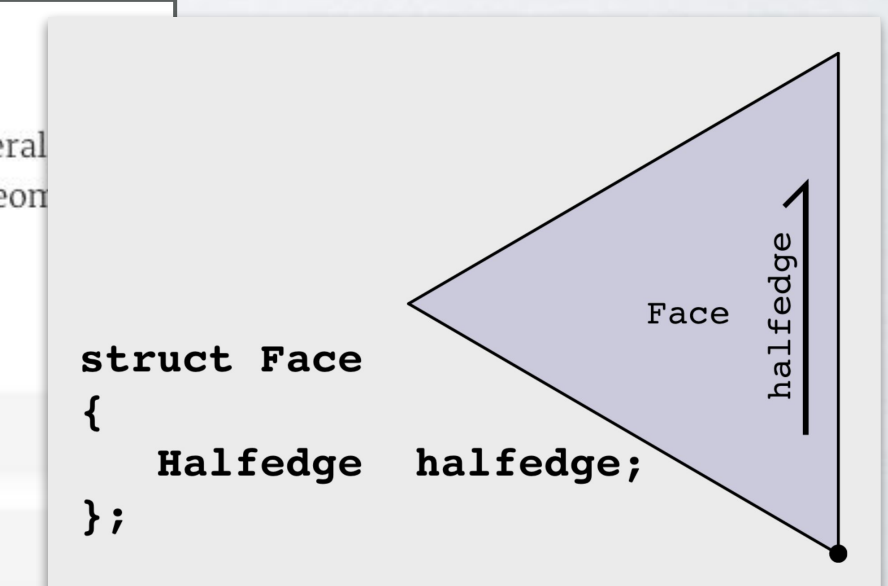
Face

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 geometry 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 $N \times N$ identity matrix of the requested type.

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

Shift the diagonal of matrix, by adding $A + \text{shiftDiagonal} * \text{identityMatrix}()$.

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

- 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

```
// 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 **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.

 `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 internally.

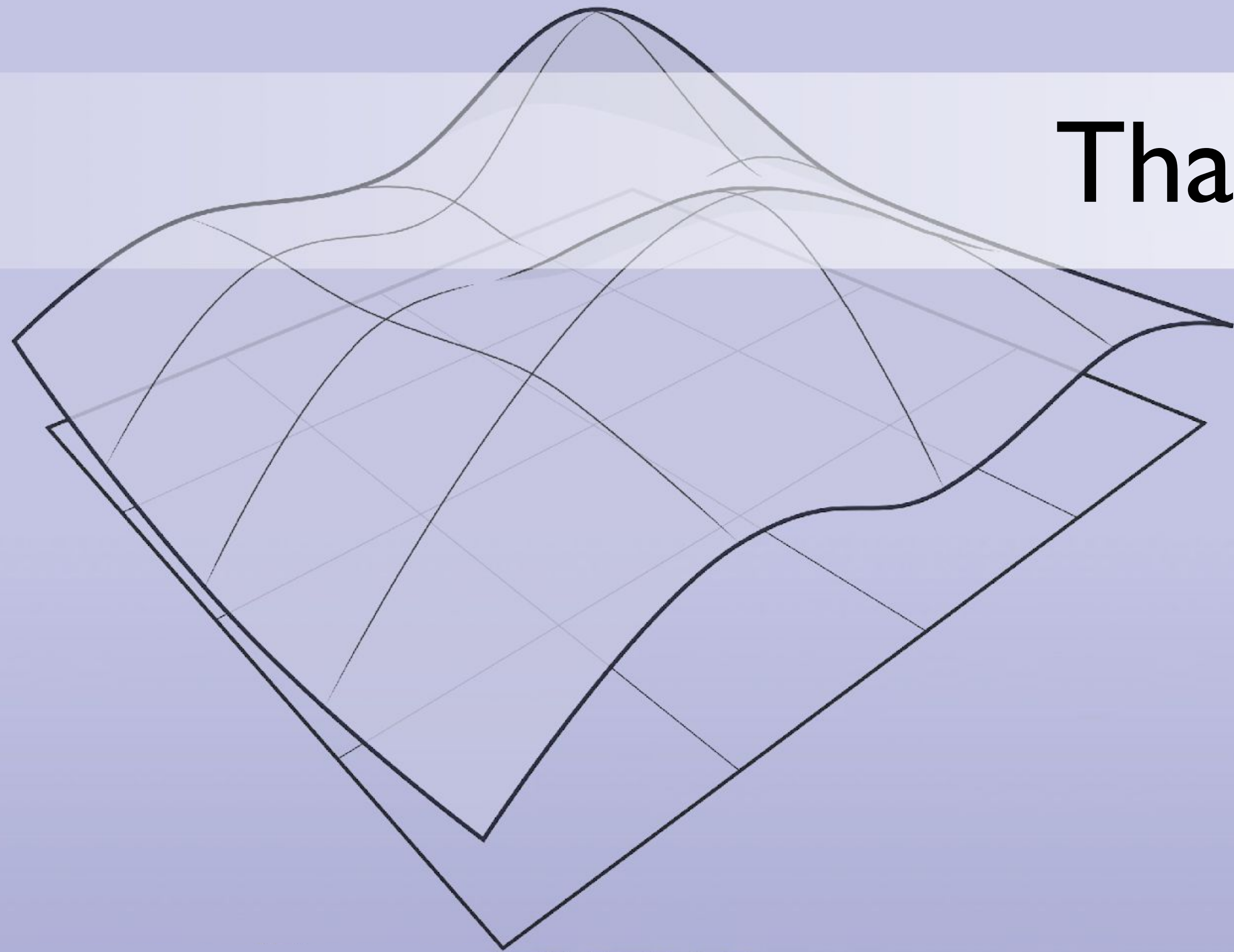
 `Vector<T> solvePositiveDefinite(SparseMatrix<T>& matrix, const Vector<T>& rhs)` 

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

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



Thanks!



**Discrete Differential
Geometry:
An Applied Introduction**