Lecture 12

- Applications of a Hash Table
  - Vertex-to-polygon table
  - Edte-to-polygon table
- Polygonal Mesh Data Structure
-

An application of the hash table

- Typical engineering-computation problem becomes a massive book-keeping problem.
  - Mesh generation
  - Finite-element analysis
  - Facet repair
  - Feature identification
  - Path finding
  - etc.

An application of the hash table

- Quickly finding polygons from a vertex.
- Quickly finding polygons from an edge (pair of vertices)
- Quickly finding a neighboring polygon

Example: Vertex-to-polygon table

- Hash key = Hash code = Vertex search key
- Value = std::vector of YsShell::PolygonHandle s

vertex_to_polygon.h

```
#ifndef VERTEX_TO_POLYGON_IS_INCLUDED
#define VERTEX_TO_POLYGON_IS_INCLUDED

#include <vector>
#include <ysshellext.h>
#include "hashtable.h"

template <>
inline unsigned long long int HashTable <YSHASHKEY,std::vector <YsShell::PolygonHandle> >::
    HashCode(const YSHASHKEY &key) const
{
    return key;
}

class VertexToPolygonTable : public HashTable <YSHASHKEY,std::vector <YsShell::PolygonHandle> >
{
public:
    void SetShell(const YsShellExt &shl);
    const std::vector <YsShell::PolygonHandle> *FindPolygonFromVertex(
        const YsShellExt &shl,
        YsShell::VertexHandle vtHd) const;
};

#endif
```

vertex_to_polygon.cpp

```
#include "vertex_to_polygon.h"


void VertexToPolygonTable::SetShell(const YsShellExt &shl)
{
    CleanUp();
    Resize(shl.GetNumPolygon());

    for(auto plHd : shl.AllPolygon())
    {
        for(auto vtHd : shl.GetPolygonVertex(plHd))
        {
            auto hashKey=shl.GetSearchKey(vtHd);
            auto vtPlPtr=(*this)[hashKey];
            if(nullptr!=vtPlPtr)
            {
                vtPlPtr->push_back(plHd);
            }
            else
            {
                std::vector <YsShell::PolygonHandle> vtPl;
                vtPl.push_back(plHd);
                this->Update(hashKey,vtPl);
            }
        }
    }
}

const std::vector <YsShell::PolygonHandle> *VertexToPolygonTable::FindPolygonFromVertex(
    const YsShellExt &shl,
    YsShell::VertexHandle vtHd) const
{
    auto hashKey=shl.GetSearchKey(vtHd);
    return (*this)[hashKey];
}
```

Example: Edge-to-polygon hash table

- Hash key = two vertices (Vertex search keys)
- Hash code = can be anything calculated from two vertices. Eg. Sum of two search keys.
- Value = std::vector of YsShell::PolygonHandle s

edge_to_polygon.h

```
#ifndef EDGE_TO_POLYGON_IS_INCLUDED
#define EDGE_TO_POLYGON_IS_INCLUDED

#include <vector>
#include <ysshellext.h>
#include "hashtable.h"

class ShellEdgeKey
{
public:
    YSHASHKEY edVtKey[2]; // Edge Vertex Keys

    bool operator==(const ShellEdgeKey &incoming) const
    {
        return (edVtKey[0]==incoming.edVtKey[0] && edVtKey[1]==incoming.edVtKey[1]) ||
               (edVtKey[0]==incoming.edVtKey[1] && edVtKey[1]==incoming.edVtKey[0]);
    }
    bool operator!=(const ShellEdgeKey &incoming) const
    {
        return (edVtKey[0]!=incoming.edVtKey[0] || edVtKey[1]!=incoming.edVtKey[1]) &&
               (edVtKey[0]!=incoming.edVtKey[1] || edVtKey[1]!=incoming.edVtKey[0]);
    }
};

template <>
inline unsigned long long int
    HashTable<ShellEdgeKey,std::vector <YsShell::PolygonHandle> >::
        HashCode(const ShellEdgeKey &key) const
{
    return key.edVtKey[0]+key.edVtKey[1];
}

class EdgeToPolygonTable : public HashTable <ShellEdgeKey,std::vector <YsShell::PolygonHandle> >
{
public:
    void SetShell(const YsShellExt &shl);
    const std::vector <YsShell::PolygonHandle> *FindPolygonFromEdge(
        const YsShellExt &shl,
        YsShell::VertexHandle edVtHd0,YsShell::VertexHandle edVtHd1) const;
};
#endif
```

edge_to_polygon.cpp

```
#include "edge_to_polygon.h"

void EdgeToPolygonTable::SetShell(const YsShellExt &shl)
{
    CleanUp();
    Resize(shl.GetNumPolygon());

    for(auto plHd : shl.AllPolygon())
    {
        auto plVtHd=shl.GetPolygonVertex(plHd);

        for(int edIdx=0; edIdx<plVtHd.GetN(); ++edIdx)
        {
            ShellEdgeKey hashKey;
            hashKey.edVtKey[0]=shl.GetSearchKey(plVtHd[edIdx]);
            hashKey.edVtKey[1]=shl.GetSearchKey(plVtHd.GetCyclic(edIdx+1));

            auto edPlPtr=(*this)[hashKey];
            if(nullptr!=edPlPtr)
            {
                edPlPtr->push_back(plHd);
            }
            else
            {
                std::vector <YsShell::PolygonHandle> edPl;
                edPl.push_back(plHd);
                this->Update(hashKey,edPl);
            }
        }
    }
}

const std::vector <YsShell::PolygonHandle> *EdgeToPolygonTable::FindPolygonFromEdge(
        const YsShellExt &shl,
        YsShell::VertexHandle edVtHd0,YsShell::VertexHandle edVtHd1) const
{
    ShellEdgeKey hashKey;
    hashKey.edVtKey[0]=shl.GetSearchKey(edVtHd0);
    hashKey.edVtKey[1]=shl.GetSearchKey(edVtHd1);
    return (*this)[hashKey];
}
```

Set up the table roughly good size for the number of polygons assuming most of them are triangles.

For each polygon, each edge

If the edge is already in the table, add the polygon. If not, make a new array for the edge and add.

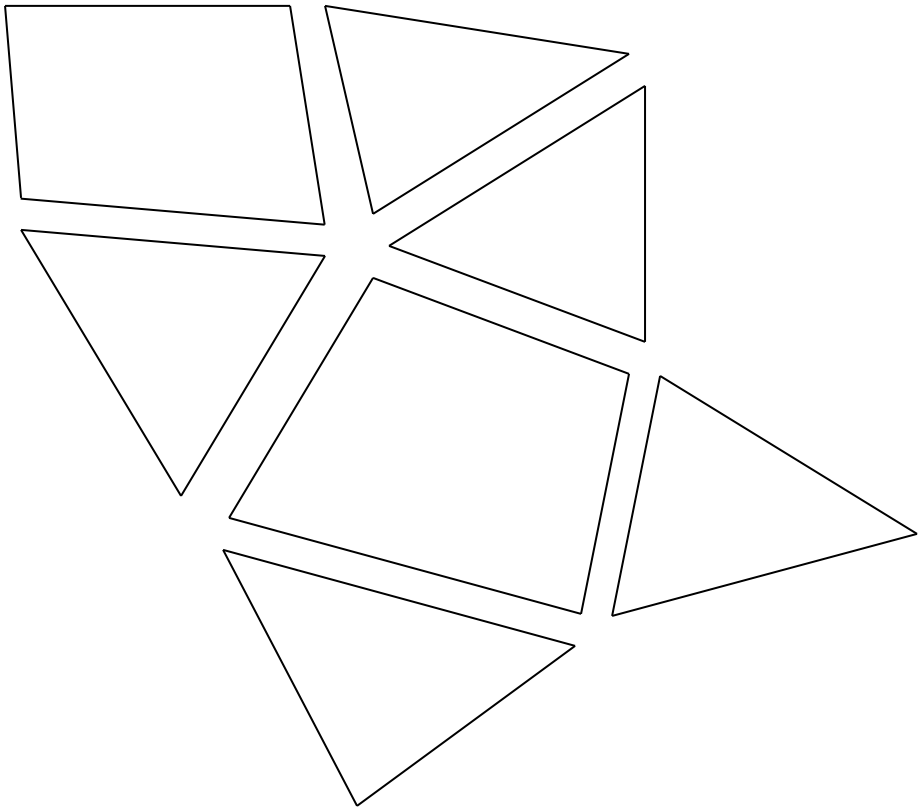
Finding is this easy.

For practical purposes….

- Such hash tables must be in sync with the editing operations.
- YsShellExt has its own hash table internally, and can be turned on by calling EnableSearch().

Polygonal Mesh

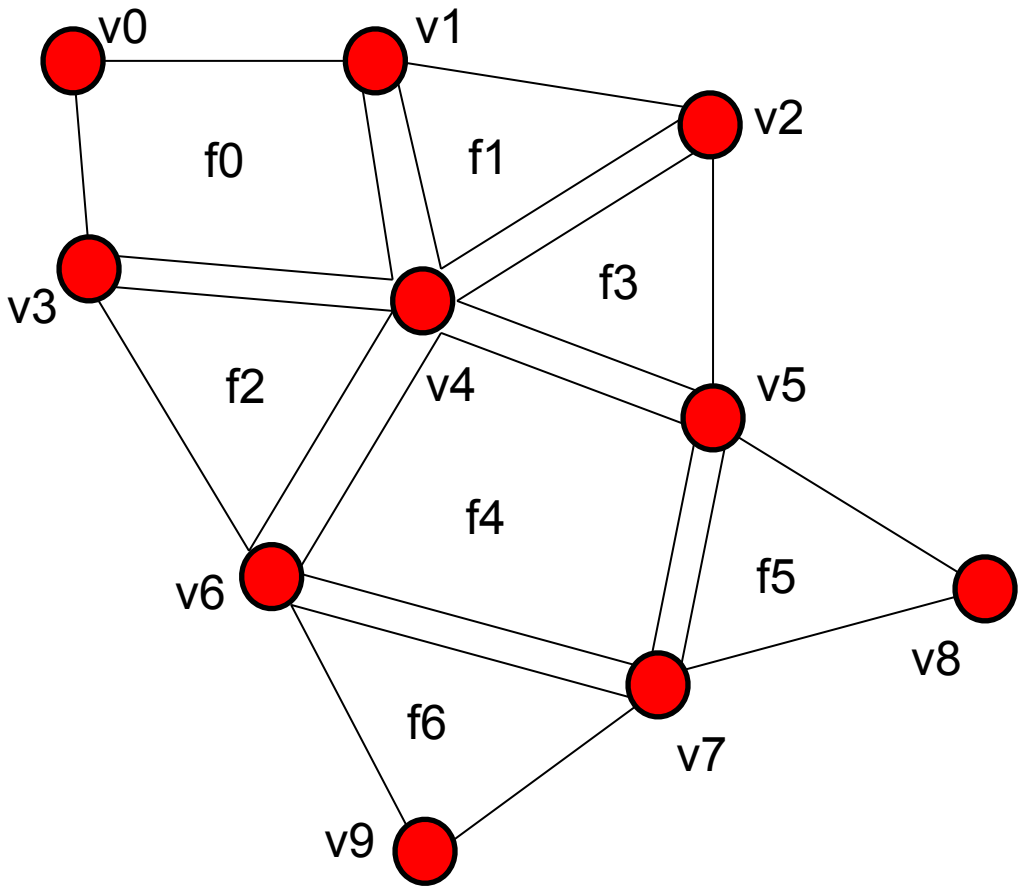
- Minimum information is a set of disconnected polygons (eg. STL data)
- Minimum information is good for visualizing.
- Useless for other purposes.



Better-Than Minimum Information

- Vertices & Connections
- A list of vertices
- A list of polygons, each of which has a chain of vertices

f0 {v1, v0, v3, v4}
f1 {v2, v1, v4}
f2 {v4, v3, v6}
f3 {v2, v4, v5}
f4 {v5, v4, v6, v7}
f5 {v7, v8, v5}
f6 {v9, v7, v6}

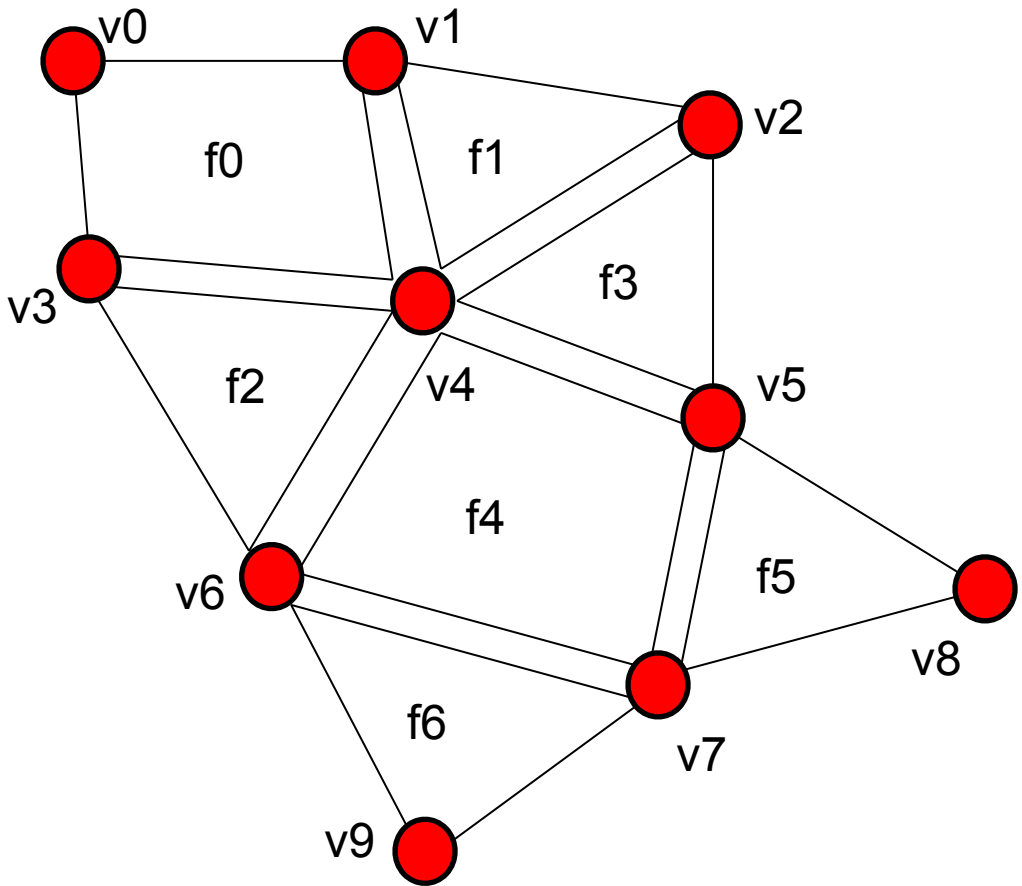


More useful information

- Polygons are seach-able from a vertex or an edge.
- Can be done by keeping a hash table.

v0 {f0}
v1 {f0, f1}
v2 {f1, f3}
v3 {f0, f2}
v4 {f0, f1, f2, f3, f4}
v5 {f3, f4, f5}
:

e(v0,v1) {f0}
e(v1,v2) {f1}
e(v1,v4) {f0, f1}
e(v3,v4) {f0, f2}
e(v5,v7) {f4, f5}
:



Richer information

- Constraint edges
  - Also called feature edges
  - Chain of vertices
  - Automatic detection is still a hot research topic
- Face groups
  - Also called segments
  - Group of polygons
  - In many cases dual of constraint edges (constraint edges are the boundaries between face groups)
  - Automatic segmentation that works for general geometry does not exist yet.