A Triangle Mesh Data Structure

David Eberly
Magic Software, Inc.
6006 Meadow Run Court
Chapel Hill, NC 27516
eberly@magic-software.com

1 Introduction

In many computer graphics applications it is necessary to manage a mesh of triangles. The mesh components are vertices, edges, and triangles. An application might require knowledge of the various connections between the mesh components. These connections can be managed independently of the actual vertex positions. This document describes a simple data structure that is convenient for managing the connections. This is not the only possible data structure. Many other types exist and have support for various queries about meshes.

2 The Data Structure

The data structure representing the mesh provides support for two basic operations, inserting triangles and removing triangles. It also supports an edge collapse operation that is useful in triangle decimation schemes. The structure provides no support for the vertex positions, but it does assume that each vertex is assigned a unique integer identifier, typically the index of that vertex in an array of contiguous vertex positions. A *mesh vertex* is defined by a single integer and is denoted by $\langle v \rangle$. A *mesh edge* is defined by a pair of integers $\langle v_0, v_1 \rangle$, each integer corresponding to an end point of the edge. To support edge maps, the edges are stored so that $v_0 = \min(v_0, v_1)$. A *triangle component* is defined by a triple of integers $\langle v_0, v_1, v_2 \rangle$, each integer corresponding to a vertex of the triangle. To support triangle maps, the triangles are stored so that $v_0 = \min(v_0, v_1, v_2)$. Observe that $\langle v_0, v_1, v_2 \rangle$ and $\langle v_0, v_2, v_1 \rangle$ are treated as different triangles. An application requiring double–sided triangles must insert both triples into the data structure. For the sake of avoiding constant reminders about order of indices, in the remainder of the document the pair/triple information does not imply the vertices are ordering in any way (although the implementation does handle the ordering).

Connectivity between the components is completely determined by the set of triples representing the triangles. A triangle $t = \langle v_0, v_1, v_2 \rangle$ has vertices v_0, v_1, and v_2. It has edges $e_0 = \langle v_0, v_1 \rangle$, $e_1 = \langle v_1, v_2 \rangle$, and $e_2 = \langle v_2, v_0 \rangle$. The inverse connections are also known. Vertex v_0 is adjacent to edges e_0 and e_2 and to triangle t. Vertex v_1 is adjacent to edges e_0 and e_1 and to triangle t. Vertex v_2 is adjacent to edges e_1 and e_2 and to triangle t. All three edges e_0, e_1, and e_2 are adjacent to t.

How much of this information a data structure stores is dependent on the needs of an application. Moreover, the application might want to have additional information stored at the components. The information stored at a vertex, edge, or triangle is referred to as the *vertex attribute*, *edge attribute*, or *triangle attribute*. The abstract representations of these for the simple data structure described here are

```
    Vertex = <integer>;  // v
    Edge = <integer,integer>;  // v0, v1
    Triangle <integer,integer,integer>;  // v0, v1, v2

    VData = <application-specific vertex data>;
    EData = <application-specific edge data>;
    TData = <application-specific triangle data>;

    VAttribute = <VData,set<Edge>,set<Triangle>>;  // data, eset, tset
    EAttribute = <EData,set<Triangle>>;  // data, tset
    TAttribute = <TData>;  // data

    VPair = pair<Vertex,VAttribute>;
    EPair = pair<Edge,EAttribute>;
    TPair = pair<Triangle,TAttribute>;

    VMap = map<VPair>;
    EMap = map<EPair>;
    TMap = map<TPair>;

    Mesh = <VMap,EMap,TMap>; // vmap, emap, tmap
```

The maps support the standard insertion and removal functions for a hash table. Insertion occurs only if the item does not already exist. Removal occurs only if the item does exist.

3 Insertion of a Triangle

Given a triangle $\langle v_0, v_1, v_2\rangle$, the insertion operation is denoted `InsertTriangle(v0,v1,v2)`. On the very first insertion, three vertices, three edges, and a triangle are added to the data structure. All of these components are created during the call. On subsequent insertions, some mesh components might already exist and need not be created. It is useful to provide a callback mechanism through which the application is informed when a component is created. Pseudocode for insertion is shown below. The callbacks are denoted `On[Foo]Create`, take as input the appropriate mesh component, and return the appropriate type data object.

```
void InsertTriangle (int v0, int v1, int v2)
{
    Triangle t(v0,v1,v2);
    Edge e0(v0,v1), e1(v1,v2), e2(v2,v0);

    // insert vertices
    for (i = 0; i < 3; i++)
    {
        <VPair,bool> (p,inserted) = vmap.insert(VPair(v[i],VAttribute()));
        if ( inserted ) p.VAttribute.data = OnVertexCreate(v[i]);
        p.VAttribute.eset.insert(e[i]);
        p.VAttribute.eset.insert(e[(i+2) mod 3]);
        p.VAttribute.tset.insert(t);
    }

    // insert edges
    for (i = 0; i < 3; i++)
    {
        <EPair,bool> (p,inserted) = emap.insert(EPair(e[i],EAttribute()));
        if ( inserted ) p.EAttribute.data = OnEdgeCreate(e[i]);
        p.EAttribute.tset.insert(t);
    }

    // insert triangle
    <TPair,bool> (p,inserted) = tmap.insert(TPair(t,TAttribute()));
    if ( inserted ) p.TAttribute.data = OnTriangleCreate(t);
}
```

4 Removal of a Triangle

Given a triangle $\langle v_0, v_1, v_2 \rangle$, the removal operation is denoted `RemoveTriangle(v0,v1,v2)`. On removal, some mesh components are modified, yet continue to exist because they are shared by other triangles. Once the last reference to a component is eliminated, only then is the component destroyed. It is useful to provide a callback mechanism through which the application is informed when a component is destroyed. Pseudocode for removal is shown below. The callbacks are denoted `On[Foo]Destroy` and take as input the appropriate vertex indices and the data associated with the to–be–destroyed component.

```
void RemoveTriangle (int v0, int v1, int v2)
{
    Triangle t(v0,v1,v2);
    Edge e0(v0,v1), e1(v1,v2), e2(v2,v0);

    // remove triangle
    <TPair,bool> (p,found) = tmap.find(t);
    if ( not found )  return;
    OnTriangleDestroy(v0,v1,v2,p.TriangleAttribute.data);
    tmap.erase(t);

    // remove edges
    for (i = 0; i < 3; i++)
    {
        <EPair,bool> (p,found) = emap.find(e[i]);  // found == true;
        p.EAttribute.tset.erase(t);
        if ( empty(p.EAttribute.tset) )
        {
            OnEdgeDestroy(e[i],p.EAttribute.data);
            emap.erase(e[i]);
        }
    }

    // remove vertices
    for (i = 0; i < 3; i++)
    {
        <VPair,bool> (p,found) = vmap.find(v[i]);  // found == true
        p.VAttribute.eset.erase(e[i]);
        p.VAttribute.eset.erase(e[(i+1) mod 3]);
        p.VAttribute.tset.erase(t);
        if ( empty(p.VAttribute.eset) && empty(p.VAttribute.tset) )
        {
            OnVertexDestroy(v[i],p.VAttribute.data);
            vmap.erase(v[i]);
        }
    }
}
```

5 Edge Collapse

This operation involves identifying an edge $\langle v_k, v_t \rangle$ where v_k is called the *keep* vertex and v_t is called the *throw* vertex. The triangles that share this edge are removed from the mesh. The vertex v_t is also removed from the mesh. Any triangles that shared v_t have that vertex replaced by v_k. Figure 1 shows a triangle mesh and a sequence of three edge collapses applied to the mesh.

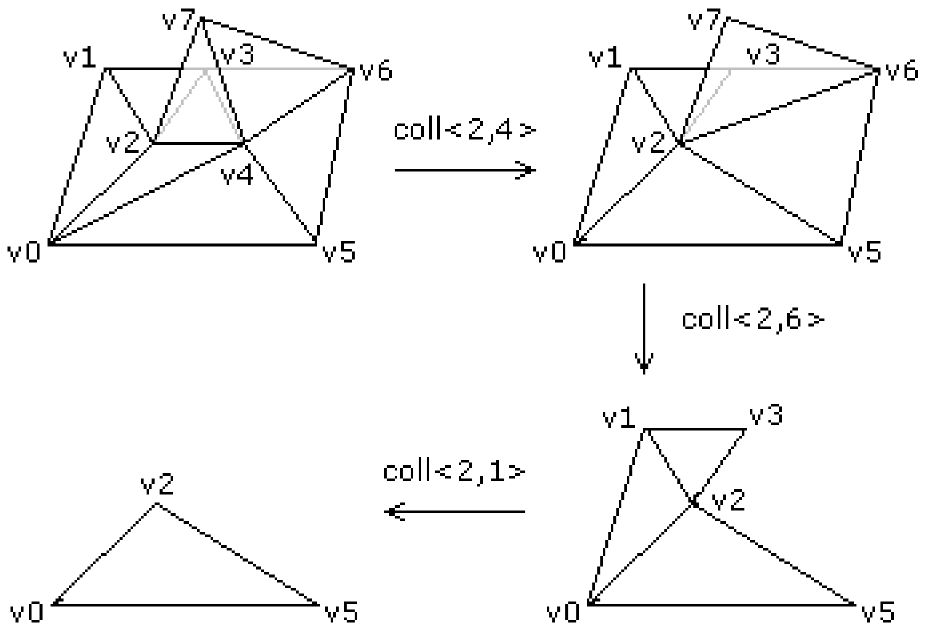

Figure 1. A triangle mesh with three edge collapses applied to it.

The operation is denoted `CollapseEdge(vk,vt)`. Pseudocode is shown below.

```
void CollapseEdge (int vk, int vt)
{
    <EPair,bool> (p,found) = emap.find(Edge(vk,vt));
    if ( not found ) return;

    // remove the triangles sharing the edge
    for ( each t in p.EAttribute.tset ) do
        RemoveTriangle(t.v0,t.v1,t.v2);

    // Replace ‘throw’ vertices by ‘keep’ vertices in the remaining
    // triangles at the ‘throw’ vertex.  The old triangles are removed
    // and the modified triangles are inserted.
    <VPair,bool> (p,found) = vmap.find(vt);
    if ( found )
    {
        for ( each t in p.VAttribute.tset )
        {
            RemoveTriangle(t.v0,t.v1,t.v2);
            for (i = 0; i < 3; i++)
            {
                if ( t.v[i] == vt )
                {
                    t.v[i] = vk;
                    break;
                }
            }
            InsertTriangle(t.v0,t.v1,t.v2);
        }
    }

    // Some edges of the triangles at ‘keep’ might have been removed
    // because they were shared by the ‘throw’ vertex.  Reinsert the
    // triangles.
    <VPair,bool> (p,found) = vmap.find(vk);
    if ( found )
    {
        for ( each t in p.VAttribute.tset )
            InsertTriangle(t.v0,t.v1,t.v2);
    }
}
```

6 The Implementation

Files `MgcMesh.h`, `MgcMesh.inl`, and `MgcMesh.cpp` contain an implementation of the data structure described in this document. The file `TestMesh.cpp` constructs the triangle mesh shown in Figure 1 and performs the edge collapses. This file contains a Mesh–derived class that overrides the creation and destruction callbacks. Those callbacks just print to a file what events are occurring during the program run. The output files are provided at the end of the document.

6.1 MgcMesh.h

```
// Magic Software, Inc.
// http://www.magic-software.com
// Copyright (c) 2000, 2001.  All Rights Reserved
//
// Source code from Magic Software is supplied under the terms of a license
// agreement and may not be copied or disclosed except in accordance with the
// terms of that agreement.  The various license agreements may be found at
// the Magic Software web site.  This file is subject to the license
//
// FREE SOURCE CODE
// http://www.magic-software.com/License/free.pdf

#ifndef MGCMESH_H
#define MGCMESH_H

#include <map>
#include <set>
using namespace std;


namespace Mgc {

class Mesh
{
public:
    Mesh ();
    virtual ~Mesh ();

    // Insert and remove triangles.  The functions are virtual in case a
    // derived class wants to implement pre- and post-operation semantics.
    virtual void InsertTriangle (int iV0, int iV1, int iV2);
    virtual void RemoveTriangle (int iV0, int iV1, int iV2);

    // Remove the edge <vkeep,vthrow> from the mesh.  All triangles sharing
    // vertex <vthrow> have it replaced by vertex <vkeep>.  The function is
    // virtual in case a derived class wants to implement pre- and
    // post-collapse semantics.
    virtual void CollapseEdge (int iVKeep, int iVThrow);

    // write the mesh to an ASCII file
```

```
void Print (const char* acFilename);


// vertex is <v>
// edge is <v0,v1> where v0 = min(v0,v1)
// triangle is <v0,v1,v2> where v0 = min(v0,v1,v2)

class Edge
{
public:
    Edge (int iV0, int iV1);
    bool operator< (const Edge& rkE) const;
    bool operator== (const Edge& rkE) const;
    bool operator!= (const Edge& rkE) const;

    int m_aiV[2];
};

class Triangle
{
public:
    Triangle (int iV0, int iV1, int iV2);
    bool operator< (const Triangle& rkT) const;

    int m_aiV[3];
};

// Callbacks that are executed when vertices, edges, or triangles are
// created or deleted during triangle insertion, triangle removal, or
// edge collapsing.  The default behavior for the creation callbacks is
// to return NULL pointers.  A derived class may override the creation
// callbacks and return data to be attached to the specific mesh
// component.  The default behavior for the destruction callbacks is to
// do nothing.  A derived class may override the destruction callbacks
// and handle the data that was detached from the specific mesh component
// before its destruction.
virtual void* OnVertexCreate (int iV);
virtual void OnVertexDestroy (int iV, void* pvData);
virtual void* OnEdgeCreate (const Edge& rkE);
virtual void OnEdgeDestroy (const Edge& rkE, void* pvData);
virtual void* OnTriangleCreate (const Triangle& rkT);
virtual void OnTriangleDestroy (const Triangle& rkT, void* pvData);

// vertex attributes
void* GetData (int iV);
set<Edge> GetEdges (int iV);
set<Triangle> GetTriangles (int iV);

// edge attributes
void* GetData (int iV0, int iV1);
set<Triangle> GetTriangles (int iV0, int iV1);
```

```
    // triangle attributes
    void* GetData (int iV0, int iV1, int iV2);

private:
    class VertexAttribute
    {
    public:
        void* m_pvData;
        set<Edge> m_kESet;
        set<Triangle> m_kTSet;
    };

    class EdgeAttribute
    {
    public:
        void* m_pvData;
        set<Triangle> m_kTSet;
    };

    class TriangleAttribute
    {
    public:
        void* m_pvData;
    };

    map<int,VertexAttribute> m_kVMap;
    map<Edge,EdgeAttribute> m_kEMap;
    map<Triangle,TriangleAttribute> m_kTMap;
};

#include "MgcMesh.inl"

} // namespace Mgc

#endif
```

6.2 MgcMesh.inl

```
// Magic Software, Inc.
// http://www.magic-software.com
// Copyright (c) 2000, 2001.  All Rights Reserved
//
// Source code from Magic Software is supplied under the terms of a license
// agreement and may not be copied or disclosed except in accordance with the
// terms of that agreement.  The various license agreements may be found at
// the Magic Software web site.  This file is subject to the license
//
// FREE SOURCE CODE
// http://www.magic-software.com/License/free.pdf

//----------------------------------------------------------------------------
inline void* Mesh::OnVertexCreate (int)
{
    return NULL;
}
//----------------------------------------------------------------------------
inline void Mesh::OnVertexDestroy (int,void*)
{
}
//----------------------------------------------------------------------------
inline void* Mesh::OnEdgeCreate (const Edge&)
{
    return NULL;
}
//----------------------------------------------------------------------------
inline void Mesh::OnEdgeDestroy (const Edge&,void*)
{
}
//----------------------------------------------------------------------------
inline void* Mesh::OnTriangleCreate (const Triangle&)
{
    return NULL;
}
//----------------------------------------------------------------------------
inline void Mesh::OnTriangleDestroy (const Triangle&,void*)
{
}
//----------------------------------------------------------------------------
inline bool Mesh::Edge::operator== (const Edge& rkE) const
{
    return m_aiV[0] == rkE.m_aiV[0] && m_aiV[1] == rkE.m_aiV[1];
}
//----------------------------------------------------------------------------
inline bool Mesh::Edge::operator!= (const Edge& rkE) const
{
    return m_aiV[0] != rkE.m_aiV[0] || m_aiV[1] != rkE.m_aiV[1];
}
//----------------------------------------------------------------------------
```

6.3 MgcMesh.cpp

```
// Magic Software, Inc.
// http://www.magic-software.com
// Copyright (c) 2000, 2001.  All Rights Reserved
//
// Source code from Magic Software is supplied under the terms of a license
// agreement and may not be copied or disclosed except in accordance with the
// terms of that agreement.  The various license agreements may be found at
// the Magic Software web site.  This file is subject to the license
//
// FREE SOURCE CODE
// http://www.magic-software.com/License/free.pdf

#ifdef WIN32
// Disable the warning about truncating the debug names to 255 characters.
// This warning shows up often with STL code.
#pragma warning( disable : 4786 )
#endif

#include <cassert>
#include <fstream>
#include "MgcMesh.h"

using namespace Mgc;

//----------------------------------------------------------------------------
Mesh::Mesh ()
{
}
//----------------------------------------------------------------------------
Mesh::~Mesh ()
{
    map<int,VertexAttribute>::iterator pkV;
    for (pkV = m_kVMap.begin(); pkV != m_kVMap.end(); pkV++)
        OnVertexDestroy(pkV->first,pkV->second.m_pvData);

    map<Edge,EdgeAttribute>::iterator pkE;
    for (pkE = m_kEMap.begin(); pkE != m_kEMap.end(); pkE++)
        OnEdgeDestroy(pkE->first,pkE->second.m_pvData);

    map<Triangle,TriangleAttribute>::iterator pkT;
    for (pkT = m_kTMap.begin(); pkT != m_kTMap.end(); pkT++)
        OnTriangleDestroy(pkT->first,pkT->second.m_pvData);
}
//----------------------------------------------------------------------------
void Mesh::InsertTriangle (int iV0, int iV1, int iV2)
{
    // attempt to insert triangle
    Triangle kT(iV0,iV1,iV2);
    pair<map<Triangle,TriangleAttribute>::iterator,bool> kRT =
        m_kTMap.insert(make_pair(kT,TriangleAttribute()));
```

```
if ( kRT.second == true )
{
    // triangle insertion occurred
    kRT.first->second.m_pvData = OnTriangleCreate(kT);
}

Edge kE0(iV0,iV1), kE1(iV1,iV2), kE2(iV2,iV0);

// attempt to insert vertices
pair<map<int,VertexAttribute>::iterator,bool> kRV;

kRV = m_kVMap.insert(make_pair(iV0,VertexAttribute()));
if ( kRV.second == true )
{
    // vertex insertion occurred
    kRV.first->second.m_pvData = OnVertexCreate(iV0);
}
kRV.first->second.m_kESet.insert(kE0);
kRV.first->second.m_kESet.insert(kE2);
kRV.first->second.m_kTSet.insert(kT);

kRV = m_kVMap.insert(make_pair(iV1,VertexAttribute()));
if ( kRV.second == true )
{
    // vertex insertion occurred
    kRV.first->second.m_pvData = OnVertexCreate(iV1);
}
kRV.first->second.m_kESet.insert(kE0);
kRV.first->second.m_kESet.insert(kE1);
kRV.first->second.m_kTSet.insert(kT);

kRV = m_kVMap.insert(make_pair(iV2,VertexAttribute()));
if ( kRV.second == true )
{
    // vertex insertion occurred
    kRV.first->second.m_pvData = OnVertexCreate(iV2);
}
kRV.first->second.m_kESet.insert(kE1);
kRV.first->second.m_kESet.insert(kE2);
kRV.first->second.m_kTSet.insert(kT);

// attempt to insert edges
pair<map<Edge,EdgeAttribute>::iterator,bool> kRE;

kRE = m_kEMap.insert(make_pair(kE0,EdgeAttribute()));
if ( kRE.second == true )
{
    // edge insertion occurred
    kRE.first->second.m_pvData = OnEdgeCreate(kE0);
}
kRE.first->second.m_kTSet.insert(kT);
```

```
    kRE = m_kEMap.insert(make_pair(kE1,EdgeAttribute()));
    if ( kRE.second == true )
    {
        // edge insertion occurred
        kRE.first->second.m_pvData = OnEdgeCreate(kE1);
    }
    kRE.first->second.m_kTSet.insert(kT);

    kRE = m_kEMap.insert(make_pair(kE2,EdgeAttribute()));
    if ( kRE.second == true )
    {
        // edge insertion occurred
        kRE.first->second.m_pvData = OnEdgeCreate(kE2);
    }
    kRE.first->second.m_kTSet.insert(kT);
}
//----------------------------------------------------------------------------
void Mesh::RemoveTriangle (int iV0, int iV1, int iV2)
{
    // attempt to remove triangle
    Triangle kT(iV0,iV1,iV2);
    map<Triangle,TriangleAttribute>::iterator pkT = m_kTMap.find(kT);
    if ( pkT == m_kTMap.end() )
    {
        // triangle does not exist, nothing to do
        return;
    }

    // triangle exists, remove it
    OnTriangleDestroy(kT,pkT->second.m_pvData);
    m_kTMap.erase(kT);

    Edge kE0(iV0,iV1), kE1(iV1,iV2), kE2(iV2,iV0);

    // update vertices
    map<int,VertexAttribute>::iterator pkV;

    pkV = m_kVMap.find(iV0);
    assert( pkV != m_kVMap.end() );
    pkV->second.m_kESet.erase(kE0);
    pkV->second.m_kESet.erase(kE2);
    pkV->second.m_kTSet.erase(kT);
    if ( pkV->second.m_kESet.size() == 0
    &&   pkV->second.m_kTSet.size() == 0 )
    {
        // last reference to v0
        OnVertexDestroy(iV0,pkV->second.m_pvData);
        m_kVMap.erase(iV0);
    }

    pkV = m_kVMap.find(iV1);
    assert( pkV != m_kVMap.end() );
```

```
pkV->second.m_kESet.erase(kE0);
pkV->second.m_kESet.erase(kE1);
pkV->second.m_kTSet.erase(kT);
if ( pkV->second.m_kESet.size() == 0
&&   pkV->second.m_kTSet.size() == 0 )
{
    // last reference to v1
    OnVertexDestroy(iV1,pkV->second.m_pvData);
    m_kVMap.erase(iV1);
}

pkV = m_kVMap.find(iV2);
assert( pkV != m_kVMap.end() );
pkV->second.m_kESet.erase(kE1);
pkV->second.m_kESet.erase(kE2);
pkV->second.m_kTSet.erase(kT);
if ( pkV->second.m_kESet.size() == 0
&&   pkV->second.m_kTSet.size() == 0 )
{
    // last reference to v2
    OnVertexDestroy(iV2,pkV->second.m_pvData);
    m_kVMap.erase(iV2);
}

// update edges
map<Edge,EdgeAttribute>::iterator pkE;

pkE = m_kEMap.find(kE0);
assert( pkE != m_kEMap.end() );
pkE->second.m_kTSet.erase(kT);
if ( pkE->second.m_kTSet.size() == 0 )
{
    // last reference to <v0,v1>
    OnEdgeDestroy(kE0,pkE->second.m_pvData);
    m_kEMap.erase(kE0);
}

pkE = m_kEMap.find(kE1);
assert( pkE != m_kEMap.end() );
pkE->second.m_kTSet.erase(kT);
if ( pkE->second.m_kTSet.size() == 0 )
{
    // last reference to <v1,v2>
    OnEdgeDestroy(kE1,pkE->second.m_pvData);
    m_kEMap.erase(kE1);
}

pkE = m_kEMap.find(kE2);
assert( pkE != m_kEMap.end() );
pkE->second.m_kTSet.erase(kT);
if ( pkE->second.m_kTSet.size() == 0 )
{
        // last reference to <v2,v0>
        OnEdgeDestroy(kE2,pkE->second.m_pvData);
        m_kEMap.erase(kE2);
    }
}
//----------------------------------------------------------------------------
void Mesh::CollapseEdge (int iVKeep, int iVThrow)
{
    // find the edge to collapse
    Edge kCollapse(iVKeep,iVThrow);
    map<Edge,EdgeAttribute>::iterator pkE = m_kEMap.find(kCollapse);
    if ( pkE == m_kEMap.end() )
    {
        // requested edge does not exist
        return;
    }

    // Remove the collapse-edge-shared triangles.  Copying the triangle set
    // from the collapse edge first is required since removal of the first
    // triangle will cause the collapse edge to be removed from the edge map,
    // thereby invalidating any iterator that points to data in the collapse
    // edge.
    set<Triangle> kTSet = pkE->second.m_kTSet;
    set<Triangle>::iterator pkT = kTSet.begin();
    while ( pkT != kTSet.end() )
    {
        RemoveTriangle(pkT->m_aiV[0],pkT->m_aiV[1],pkT->m_aiV[2]);
        pkT++;
    }

    // Replace 'throw' vertices by 'keep' vertices in the remaining triangles
    // at the 'throw' vertex.  The old triangles are removed and the modified
    // triangles are inserted.
    map<int,VertexAttribute>::iterator pkV = m_kVMap.find(iVThrow);
    if ( pkV != m_kVMap.end() )
    {
        kTSet = pkV->second.m_kTSet;
        pkT = kTSet.begin();
        while ( pkT != kTSet.end() )
        {
            RemoveTriangle(pkT->m_aiV[0],pkT->m_aiV[1],pkT->m_aiV[2]);
            for (int i = 0; i < 3; i++)
            {
                if ( pkT->m_aiV[i] == iVThrow )
                {
                    pkT->m_aiV[i] = iVKeep;
                    break;
                }
            }
            InsertTriangle(pkT->m_aiV[0],pkT->m_aiV[1],pkT->m_aiV[2]);
            pkT++;
        }
    }

    // Some edges of the triangles at 'keep' might have been removed because
    // they were shared by the 'throw' vertex.  Reinsert the triangles.
    pkV = m_kVMap.find(iVKeep);
    if ( pkV != m_kVMap.end() )
    {
        kTSet = pkV->second.m_kTSet;
        pkT = kTSet.begin();
        while ( pkT != kTSet.end() )
        {
            InsertTriangle(pkT->m_aiV[0],pkT->m_aiV[1],pkT->m_aiV[2]);
            pkT++;
        }
    }
}
//----------------------------------------------------------------------------
void Mesh::Print (const char* acFilename)
{
    ofstream kOStr(acFilename);

    map<int,VertexAttribute>::iterator pkV;
    map<Edge,EdgeAttribute>::iterator pkE;
    map<Triangle,TriangleAttribute>::iterator pkT;
    set<int>::iterator pkI;
    set<Edge>::iterator pkEI;
    set<Triangle>::iterator pkTI;

    kOStr << "vertex quantity = " << m_kVMap.size() << endl;
    for (pkV = m_kVMap.begin(); pkV != m_kVMap.end(); pkV++)
    {
        kOStr << "v<" << pkV->first << "> : e ";

        pkEI = pkV->second.m_kESet.begin();
        while ( pkEI != pkV->second.m_kESet.end() )
        {
            kOStr << '<' << pkEI->m_aiV[0] << ',' << pkEI->m_aiV[1] << "> ";
            pkEI++;
        }

        kOStr << ": t ";
        pkTI = pkV->second.m_kTSet.begin();
        while ( pkTI != pkV->second.m_kTSet.end() )
        {
            kOStr << '<' << pkTI->m_aiV[0] << ',' << pkTI->m_aiV[1] << ',';
            kOStr << pkTI->m_aiV[2] << "> ";
            pkTI++;
        }
        kOStr << endl;
    }
    kOStr << endl;
    kOStr << "edge quantity = " << m_kEMap.size() << endl;
    for (pkE = m_kEMap.begin(); pkE != m_kEMap.end(); pkE++)
    {
        kOStr << "e<" << pkE->first.m_aiV[0] << ',' << pkE->first.m_aiV[1];
        kOStr << "> : t ";
        pkTI = pkE->second.m_kTSet.begin();
        while ( pkTI != pkE->second.m_kTSet.end() )
        {
            kOStr << '<' << pkTI->m_aiV[0] << ',' << pkTI->m_aiV[1] << ',';
            kOStr << pkTI->m_aiV[2] << "> ";
            pkTI++;
        }
        kOStr << endl;
    }
    kOStr << endl;

    kOStr << "triangle quantity = " << m_kTMap.size() << endl;
    for (pkT = m_kTMap.begin(); pkT != m_kTMap.end(); pkT++)
    {
        kOStr << "t<" << pkT->first.m_aiV[0] << ',' << pkT->first.m_aiV[1];
        kOStr << ',' << pkT->first.m_aiV[2]  << ">" << endl;
    }
    kOStr << endl;
}
//----------------------------------------------------------------------------
void* Mesh::GetData (int iV)
{
    map<int,VertexAttribute>::iterator pkV = m_kVMap.find(iV);
    return ( pkV != m_kVMap.end() ? pkV->second.m_pvData : NULL );
}
//----------------------------------------------------------------------------
set<Mesh::Edge> Mesh::GetEdges (int iV)
{
    map<int,VertexAttribute>::iterator pkV = m_kVMap.find(iV);
    return ( pkV != m_kVMap.end() ? pkV->second.m_kESet : set<Edge>() );
}
//----------------------------------------------------------------------------
set<Mesh::Triangle> Mesh::GetTriangles (int iV)
{
    map<int,VertexAttribute>::iterator pkV = m_kVMap.find(iV);
    return ( pkV != m_kVMap.end() ? pkV->second.m_kTSet : set<Triangle>() );
}
//----------------------------------------------------------------------------
void* Mesh::GetData (int iV0, int iV1)
{
    Edge kE(iV0,iV1);
    map<Edge,EdgeAttribute>::iterator pkE = m_kEMap.find(kE);
    return ( pkE != m_kEMap.end() ? pkE->second.m_pvData : NULL );
}
//----------------------------------------------------------------------------
set<Mesh::Triangle> Mesh::GetTriangles (int iV0, int iV1)
{
    Edge kE(iV0,iV1);
    map<Edge,EdgeAttribute>::iterator pkE = m_kEMap.find(kE);
    return ( pkE != m_kEMap.end() ? pkE->second.m_kTSet : set<Triangle>() );
}
//----------------------------------------------------------------------------
void* Mesh::GetData (int iV0, int iV1, int iV2)
{
    Triangle kT(iV0,iV1,iV2);
    map<Triangle,TriangleAttribute>::iterator pkT = m_kTMap.find(kT);
    return ( pkT != m_kTMap.end() ? pkT->second.m_pvData : NULL );
}
//----------------------------------------------------------------------------
Mesh::Edge::Edge (int iV0, int iV1)
{
    if ( iV0 < iV1 )
    {
        // v0 is minimum
        m_aiV[0] = iV0;
        m_aiV[1] = iV1;
    }
    else
    {
        // vi is minimum
        m_aiV[0] = iV1;
        m_aiV[1] = iV0;
    }
}
//----------------------------------------------------------------------------
bool Mesh::Edge::operator< (const Edge& rkE) const
{
    if ( m_aiV[1] < rkE.m_aiV[1] )
        return true;

    if ( m_aiV[1] == rkE.m_aiV[1] )
        return m_aiV[0] < rkE.m_aiV[0];

    return false;
}
//----------------------------------------------------------------------------
Mesh::Triangle::Triangle (int iV0, int iV1, int iV2)
{
    if ( iV0 < iV1 )
    {
        if ( iV0 < iV2 )
        {
            // v0 is minimum
            m_aiV[0] = iV0;
            m_aiV[1] = iV1;
            m_aiV[2] = iV2;
        }
        else
        {
            // v2 is minimum
            m_aiV[0] = iV2;
            m_aiV[1] = iV0;
            m_aiV[2] = iV1;
        }
    }
    else
    {
        if ( iV1 < iV2 )
        {
            // v1 is minimum
            m_aiV[0] = iV1;
            m_aiV[1] = iV2;
            m_aiV[2] = iV0;
        }
        else
        {
            // v2 is minimum
            m_aiV[0] = iV2;
            m_aiV[1] = iV0;
            m_aiV[2] = iV1;
        }
    }
}
//------------------------------------------------------------------------------
bool Mesh::Triangle::operator< (const Triangle& rkT) const
{
    if ( m_aiV[2] < rkT.m_aiV[2] )
        return true;

    if ( m_aiV[2] == rkT.m_aiV[2] )
    {
        if ( m_aiV[1] < rkT.m_aiV[1] )
            return true;

        if ( m_aiV[1] == rkT.m_aiV[1] )
            return m_aiV[0] < rkT.m_aiV[0];
    }

    return false;
}
//------------------------------------------------------------------------------
```

6.4 TestMesh.cpp

```
#ifdef WIN32
// Disable the warning about truncating the debug names to 255 characters.
// This warning shows up often with STL code.
#pragma warning( disable : 4786 )
#endif

#include <fstream>
using namespace std;

#include "MgcMesh.h"
using namespace Mgc;

ofstream gs_kOut("events.txt");

class MyMesh : public Mesh
{
public:
    MyMesh () { m_bInCollapseEdge = false; }

    virtual void* OnVertexCreate (int iV)
    {
        gs_kOut << "create v<" << iV << ">" << endl;
        return NULL;
    }

    virtual void OnVertexDestroy (int iV, void*)
    {
        gs_kOut << "destroy v<" << iV << ">" << endl;
    }

    virtual void* OnEdgeCreate (const Edge& rkE)
    {
        gs_kOut << "create e<" << rkE.m_aiV[0] << ',' << rkE.m_aiV[1];
        gs_kOut << ">" << endl;
        return NULL;
    }

    virtual void OnEdgeDestroy (const Edge& rkE, void*)
    {
        gs_kOut << "destroy e<" << rkE.m_aiV[0] << ',' << rkE.m_aiV[1];
        gs_kOut << ">" << endl;
    }

    virtual void* OnTriangleCreate (const Triangle& rkT)
    {
        gs_kOut << "create t<" << rkT.m_aiV[0] << ',' << rkT.m_aiV[1];
        gs_kOut << ',' << rkT.m_aiV[2] << ">" << endl;
        return NULL;
    }
    virtual void OnTriangleDestroy (const Triangle& rkT, void*)
    {
        gs_kOut << "destroy t<" << rkT.m_aiV[0] << ',' << rkT.m_aiV[1];
        gs_kOut << ',' << rkT.m_aiV[2] << ">" << endl;
    }

    virtual void InsertTriangle (int iV0, int iV1, int iV2)
    {
        gs_kOut << "insert triangle t<" << iV0 << ',' << iV1 << ',';
        gs_kOut << iV2 << ">" << endl;
        Mesh::InsertTriangle(iV0,iV1,iV2);
        gs_kOut << "end insert" << endl;
        if ( !m_bInCollapseEdge )
            gs_kOut << endl;
    }

    virtual void RemoveTriangle (int iV0, int iV1, int iV2)
    {
        gs_kOut << "remove triangle t<" << iV0 << ',' << iV1 << ',';
        gs_kOut << iV2 << ">" << endl;
        Mesh::RemoveTriangle(iV0,iV1,iV2);
        gs_kOut << "end remove" << endl;
        if ( !m_bInCollapseEdge )
            gs_kOut << endl;
    }

    virtual void CollapseEdge (int iVKeep, int iVThrow)
    {
        gs_kOut << "begin collapse e<" << iVKeep << ',' << iVThrow << ">";
        gs_kOut << endl;

        m_bInCollapseEdge = true;
        Mesh::CollapseEdge(iVKeep,iVThrow);
        m_bInCollapseEdge = false;

        gs_kOut << "end collapse" << endl << endl;
    }

protected:
    bool m_bInCollapseEdge;
};

int main ()
{
    MyMesh kMesh;

    kMesh.InsertTriangle(0,2,1);
    kMesh.InsertTriangle(1,2,3);
    kMesh.InsertTriangle(0,4,2);
    kMesh.InsertTriangle(2,4,3);
    kMesh.InsertTriangle(0,5,4);
    kMesh.InsertTriangle(5,6,4);
    kMesh.InsertTriangle(4,6,3);
    kMesh.InsertTriangle(2,4,7);
    kMesh.InsertTriangle(4,6,7);
    kMesh.Print("initial.txt");

    kMesh.CollapseEdge(2,4);
    kMesh.Print("collapse_2_4.txt");

    kMesh.CollapseEdge(2,6);
    kMesh.Print("collapse_2_6.txt");

    kMesh.CollapseEdge(2,1);
    kMesh.Print("collapse_2_1.txt");

    return 0;
}
```

6.5 Output initial.txt

```
vertex quantity = 8
v<0> : e <0,1> <0,2> <0,4> <0,5> : t <0,2,1> <0,4,2> <0,5,4>
v<1> : e <0,1> <1,2> <1,3> : t <0,2,1> <1,2,3>
v<2> : e <0,2> <1,2> <2,3> <2,4> <2,7> : t <0,2,1> <0,4,2> <1,2,3> <2,4,3> <2,4,7>
v<3> : e <1,3> <2,3> <3,4> <3,6> : t <1,2,3> <2,4,3> <3,4,6>
v<4> : e <0,4> <2,4> <3,4> <4,5> <4,6> <4,7> : t <0,4,2> <2,4,3> <0,5,4> <3,4,6> <4,5,6> <2,4,7> <4,6,7>
v<5> : e <0,5> <4,5> <5,6> : t <0,5,4> <4,5,6>
v<6> : e <3,6> <4,6> <5,6> <6,7> : t <3,4,6> <4,5,6> <4,6,7>
v<7> : e <2,7> <4,7> <6,7> : t <2,4,7> <4,6,7>

edge quantity = 16
e<0,1> : t <0,2,1>
e<0,2> : t <0,2,1> <0,4,2>
e<1,2> : t <0,2,1> <1,2,3>
e<1,3> : t <1,2,3>
e<2,3> : t <1,2,3> <2,4,3>
e<0,4> : t <0,4,2> <0,5,4>
e<2,4> : t <0,4,2> <2,4,3> <2,4,7>
e<3,4> : t <2,4,3> <3,4,6>
e<0,5> : t <0,5,4>
e<4,5> : t <0,5,4> <4,5,6>
e<3,6> : t <3,4,6>
e<4,6> : t <3,4,6> <4,5,6> <4,6,7>
e<5,6> : t <4,5,6>
e<2,7> : t <2,4,7>
e<4,7> : t <2,4,7> <4,6,7>
e<6,7> : t <4,6,7>

triangle quantity = 9
t<0,2,1>
t<0,4,2>
t<1,2,3>
t<2,4,3>
t<0,5,4>
t<3,4,6>
t<4,5,6>
t<2,4,7>
t<4,6,7>
```

6.6 Output collapse_2_4.txt

```
vertex quantity = 7
v<0> : e <0,1> <0,2> <0,5> : t <0,2,1> <0,5,2>
v<1> : e <0,1> <1,2> <1,3> : t <0,2,1> <1,2,3>
v<2> : e <0,2> <1,2> <2,3> <2,5> <2,6> <2,7> : t <0,2,1> <0,5,2> <1,2,3> <2,6,3> <2,5,6> <2,6,7>
v<3> : e <1,3> <2,3> <3,6> : t <1,2,3> <2,6,3>
v<5> : e <0,5> <2,5> <5,6> : t <0,5,2> <2,5,6>
v<6> : e <2,6> <3,6> <5,6> <6,7> : t <2,6,3> <2,5,6> <2,6,7>
v<7> : e <2,7> <6,7> : t <2,6,7>

edge quantity = 12
e<0,1> : t <0,2,1>
e<0,2> : t <0,2,1> <0,5,2>
e<1,2> : t <0,2,1> <1,2,3>
e<1,3> : t <1,2,3>
e<2,3> : t <1,2,3> <2,6,3>
e<0,5> : t <0,5,2>
e<2,5> : t <0,5,2> <2,5,6>
e<2,6> : t <2,6,3> <2,5,6> <2,6,7>
e<3,6> : t <2,6,3>
e<5,6> : t <2,5,6>
e<2,7> : t <2,6,7>
e<6,7> : t <2,6,7>

triangle quantity = 6
t<0,2,1>
t<0,5,2>
t<1,2,3>
t<2,6,3>
t<2,5,6>
t<2,6,7>
```

6.7 Output collapse_2_6.txt

```
vertex quantity = 5
v<0> : e <0,1> <0,2> <0,5> : t <0,2,1> <0,5,2>
v<1> : e <0,1> <1,2> <1,3> : t <0,2,1> <1,2,3>
v<2> : e <0,2> <1,2> <2,3> <2,5> : t <0,2,1> <0,5,2> <1,2,3>
v<3> : e <1,3> <2,3> : t <1,2,3>
v<5> : e <0,5> <2,5> : t <0,5,2>

edge quantity = 7
e<0,1> : t <0,2,1>
e<0,2> : t <0,2,1> <0,5,2>
e<1,2> : t <0,2,1> <1,2,3>
e<1,3> : t <1,2,3>
e<2,3> : t <1,2,3>
e<0,5> : t <0,5,2>
e<2,5> : t <0,5,2>

triangle quantity = 3
t<0,2,1>
t<0,5,2>
t<1,2,3>
```

6.8 Output collapse_2_1.txt

```
vertex quantity = 3
v<0> : e <0,2> <0,5> : t <0,5,2>
v<2> : e <0,2> <2,5> : t <0,5,2>
v<5> : e <0,5> <2,5> : t <0,5,2>

edge quantity = 3
e<0,2> : t <0,5,2>
e<0,5> : t <0,5,2>
e<2,5> : t <0,5,2>

triangle quantity = 1
t<0,5,2>
```

6.9 Output events.txt

```
insert triangle t<0,2,1>
create t<0,2,1>
create v<0>
create v<2>
create v<1>
create e<0,2>
create e<1,2>
create e<0,1>
end insert

insert triangle t<1,2,3>
create t<1,2,3>
create v<3>
create e<2,3>
create e<1,3>
end insert

insert triangle t<0,4,2>
create t<0,4,2>
create v<4>
create e<0,4>
create e<2,4>
end insert

insert triangle t<2,4,3>
create t<2,4,3>
create e<3,4>
end insert

insert triangle t<0,5,4>
create t<0,5,4>
create v<5>
create e<0,5>
create e<4,5>
end insert

insert triangle t<5,6,4>
create t<4,5,6>
create v<6>
create e<5,6>
create e<4,6>
end insert

insert triangle t<4,6,3>
create t<3,4,6>
create e<3,6>
end insert

insert triangle t<2,4,7>
create t<2,4,7>
create v<7>
create e<4,7>
create e<2,7>
end insert

insert triangle t<4,6,7>
create t<4,6,7>
create e<6,7>
end insert

begin collapse e<2,4>
remove triangle t<0,4,2>
destroy t<0,4,2>
end remove
remove triangle t<2,4,3>
destroy t<2,4,3>
end remove
remove triangle t<2,4,7>
destroy t<2,4,7>
destroy e<2,4>
destroy e<2,7>
end remove
remove triangle t<0,5,4>
destroy t<0,5,4>
destroy e<0,5>
destroy e<0,4>
end remove
insert triangle t<0,5,2>
create t<0,5,2>
create e<0,5>
create e<2,5>
end insert
remove triangle t<3,4,6>
destroy t<3,4,6>
destroy e<3,4>
destroy e<3,6>
end remove
insert triangle t<3,2,6>
create t<2,6,3>
create e<2,6>
create e<3,6>
end insert
remove triangle t<4,5,6>
destroy t<4,5,6>
destroy e<4,5>
destroy e<5,6>
end remove
insert triangle t<2,5,6>
create t<2,5,6>
create e<5,6>
end insert
remove triangle t<4,6,7>
```

```
destroy t<4,6,7>
destroy v<4>
destroy v<7>
destroy e<4,6>
destroy e<6,7>
destroy e<4,7>
end remove
insert triangle t<2,6,7>
create t<2,6,7>
create v<7>
create e<6,7>
create e<2,7>
end insert
insert triangle t<0,2,1>
end insert
insert triangle t<0,5,2>
end insert
insert triangle t<1,2,3>
end insert
insert triangle t<2,6,3>
end insert
insert triangle t<2,5,6>
end insert
insert triangle t<2,6,7>
end insert
end collapse

begin collapse e<2,6>
remove triangle t<2,6,3>
destroy t<2,6,3>
destroy e<3,6>
end remove
remove triangle t<2,5,6>
destroy t<2,5,6>
destroy e<5,6>
end remove
remove triangle t<2,6,7>
destroy t<2,6,7>
destroy v<6>
destroy v<7>
destroy e<2,6>
destroy e<6,7>
destroy e<2,7>
end remove
insert triangle t<0,2,1>
end insert
insert triangle t<0,5,2>
end insert
insert triangle t<1,2,3>
end insert
end collapse
```

```
begin collapse e<2,1>
remove triangle t<0,2,1>
destroy t<0,2,1>
destroy e<0,1>
end remove
remove triangle t<1,2,3>
destroy t<1,2,3>
destroy v<1>
destroy v<3>
destroy e<1,2>
destroy e<2,3>
destroy e<1,3>
end remove
insert triangle t<0,5,2>
end insert
end collapse
```