Fondamenti di Grafica Tridimensionale

Paolo Cignoni
p.cignoni@isti.cnr.it
http://vcg.isti.cnr.it/~cignoni

Simplessi

- Un ***k simplesso*** è definito come la combinazione convessa di k+1 punti non linearmente dipendenti

- k è l'ordine del simplesso
- I punti si chiamano vertici

Sotto-Simplesso

- Un simplesso σ' è detto *faccia* di un simplesso σ se e' definito da un sottoinsieme dei vertici di σ
-
- Se $\sigma \neq \sigma'$ si dice che é una faccia propria

Complesso Simpliciale

- Una collezione di simplessi Σ e' un k-complesso simpliciale se:

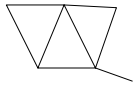
$\forall \sigma_1, \sigma_2 \in \Sigma \quad \sigma_1 \cap \sigma_2 \neq \emptyset \rightarrow \sigma_1 \cap \sigma_2$ is a simplex of Σ

$\forall \sigma \in \Sigma$ all the faces of σ belong to Σ

k is the maximum order $\forall \sigma \in \Sigma$

Complesso Simpliciale

- Un simplesso σ è massimale in un complesso simpliciale Σ se non è faccia propria di nessun altro simplesso di Σ
- Un k-complesso simpliciale Σ è massimale se tutti simplessi massimali sono di ordine k
  - In pratica non penzolano pezzi di ordine inferiore



2-Manifold

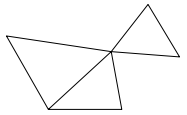
- Una superficie Σ immersa in $\boldsymbol{R^3}$ tale che ogni punto su Σ ha un intorno aperto omeomorfo ad un disco aperto o a un semidisco aperto in $\boldsymbol{R^2}$
- Esempi non manifold

Mesh

- Le classiche mesh triangolari cui siamo abituati sono 2-complessi simpliciali massimali la cui realizzazione in R3 è una superficie 2-manifold.
- Note:
  - A volte (spesso) capitano superfici non 2-manifold
  - Che siano massimali invece lo assumiamo.

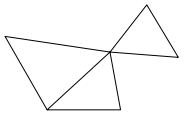
Incidenza Adiacenza

- Due simplessi σ e σ' sono incidenti se σ è una faccia propria di σ' o vale il viceversa.
- Due k-simplessi sono adiacenti se esiste un k-1 simplesso che è una faccia propria di entrambi.



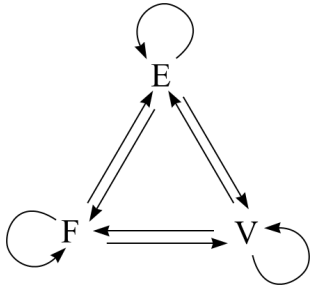
Relazioni di Adiacenza

- Per semplicità nel caso di mesh si una relazione di adiacenza con un una coppia (ordinata!) di lettere che indicano le entità coinvolte
  - FF adiacenza tra triangoli
  - FV i vertici che compongono un triangolo
  - VF i triangoli incidenti su un dato vertice



Relazioni di adiacenza

- Di tutte le possibili relazioni di adiacenza di solito vale la pena tenerne solo un sottoinsieme (su 9) e ricavare le altre proceduralmente



Relazioni di adiacenza

- In un 2-complesso simpliciale immerso in R3 (e che sia 2 manifold)
  - FV FE FF EF EV sono di card. costante
  - VV VE VF EE sono di card. variabile.
- Per risparmiare a volte si mantiene una informazione di adiacenza parziale
  - VF* memorizzo solo un riferimento dal vertice ad una delle facce e poi 'navigo' sulla mesh usando la FF per trovare le altre facce incidenti su V

Visualization and Computer Graphics Library

Visual Computing Laboratory
ISTI – CNR
Pisa

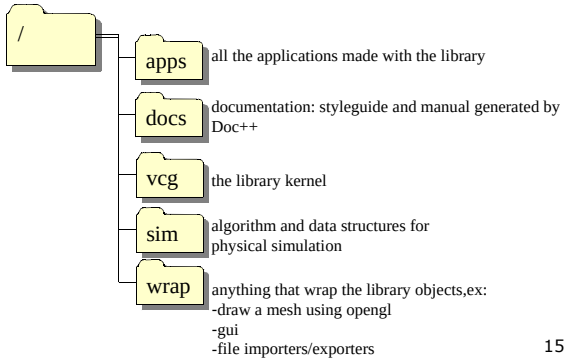
The goal

- A framework to implement algorithms on Simplicial Complexes of order d=0..3 in R^n:
  - Efficient code
  - Easy to understand
  - Flexible
  - Reusable
  - Multiplatform (MS 7.1,Intel,gnu)
  - Open Source !

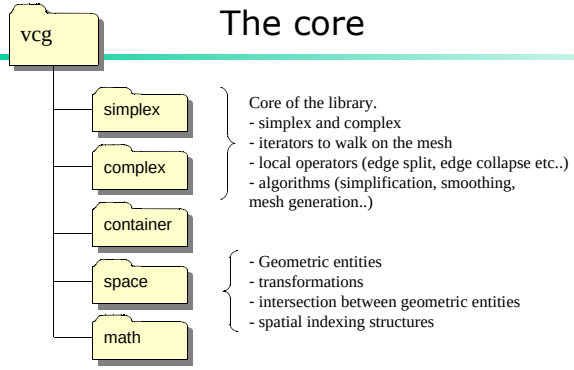
Installation and Startup

- Cvs server
  - :ext:anonymous@cvs.sourceforge.net:/cvsroot/vcg
- Download in your folder (ex: c:\base)
- Set the include path c:\base in your environment
- End of installation and startup

VCG Structure



The core



Representing Simplicial Complexes

- A good problem.
- Meshes requires different information for different algorithms and purposes
  - Topology
  - Different geometric informations
  - Additional datas
  -
- Templated solutions.
  - Generic algorithms on generic meshes

Vertex

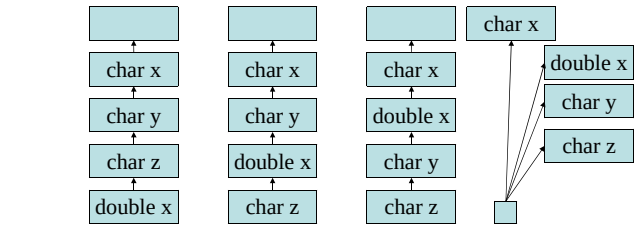
- What is a vertex?
  - position in n-space (almost always)
  - normal
  - color
  - quality
  - quadric
  - connectivity information (topology)
  - ...
- One may want any combination of attributes

Vertex (cntd)

- How to do it?
  - every user derives an empty VertexBase class to implement the vertex type
    - annoying cut and paste of code
    - need to agree upon interface (name of access functions)
    - potentially memory consuming (memory padding)
  - multiple inheritance: not well supported in old compilers. It could be done now...

memory padding

- example: compiler 8-bytes alignement (default MS compiler 7.1)
- define a structure made of 3 chars and 1 double = 11 bytes



/Zp8	16	24	32	16
/Zp4	12	16	20	12

Vertex Base Class

- VCG solution: use precompiler directives
- Code snippet taken from: vcg/simplex/vertex/base.h, NEVER included directly

```
template <class FLTYPE, class VETYPE = DUMMYEDGETYPE,
        class VFTYPE = DUMMYFACETYPE, class VTTYPE = DUMMYTETRATYPE,
        class TCTYPE = TCoord2<float,1> >
class VERTEX_TYPE
{
 ....
#ifdef __VCGLIB_VERTEX_VT
protected:
        TCTYPE _t; // <<<<== the data is defined only when needed
#endif
public:
        TCTYPE & T() // <<<<== the access function is always present
        {
#ifdef __VCGLIB_VERTEX_VT
                return _t;
#else
                assert(0);
                return *(TCTYPE*)(&_flags);
#endif
}
....
```


Includers

- A small file include base.h after defining the proper symbols
- file vcg/simplex/vertex/with/vcvn.h (*vertex with normal and color*)

```
#ifndef __VCGLIB_VERTEX__VCVN__TYPE
#define __VCGLIB_VERTEX__VCVN__TYPE

#define VERTEX_TYPE VertexVCVN

#define __VCGLIB_VERTEX_VN
#define __VCGLIB_VERTEX_VC

#include <vcg/simplex/vertex/base.h>

#undef VERTEX_TYPE
#undef __VCGLIB_VERTEX_VN
#undef __VCGLIB_VERTEX_VC

namespace vcg {

template < class VETYPE, class VFTYPE, class VTTYPE>
class VertexVCVNf : public VertexVCVN<float,VETYPE,VFTYPE,VTTYPE> {};

template < class VETYPE, class VFTYPE, class VTTYPE>
class VertexVCVNd : public VertexVCVN<double,VETYPE,VFTYPE,VTTYPE> {};

}
#endif
```


Vertex (cntd)

- Hello vertex ....
- drawbacks:
  - potentially there would be 2^*k* includer files (*k* number of attributes)
    - true, but they are written on demand
  - the base.h header file is full of directives
    - true but there is no nesting, still readable

Vertex second solution

- every attribute has its own class. Example: the normal

```
template <class T> class EmptyNormal: public T {
public:
  typedef vcg::Point3s NormalType;
  NormalType &N() { static NormalType dummy_normal(0, 0, 0); return dummy_normal;
   }
  static bool HasNormal()   { return false; }
};
template <class A, class T> class Normal: public T {
public:
  typedef A NormalType;
  NormalType &N() { return _norm; }
  static bool HasNormal()   { return true; }
private:
  NormalType _norm;
};

template <class T> class Normal3s: public Normal<vcg::Point3s, T> {};
template <class T> class Normal3f: public Normal<vcg::Point3f, T> {};
template <class T> class Normal3d: public Normal<vcg::Point3d, T> {};
```

vertex (cntd)

- All the desired attributes are passed as template class to the vertex:

```
typedef VertexSimp1< Vertex0,EdgeProto,
        vcg::vert::VFAdj,  vcg::vert::Normal3f,  vcg::vert::Color4b> MyVertex;
```

- The template parameters can be passed in any order
- More elegant
- Much more complex implementation
- Better typed: ex. the normal and the position have different type even if their structure is the very same
- Still under development...

Simplices

- Available simplices:
  - Vertex
  - Edge
  - Face
  - Tetrahedron
- A d-simplex is implemented as a collection of d+1 pointer to vertices
- They could be implemented as a single templated class

```
template <int order> Simplex{...};
```

- but they are not (at the moment)

Complexes

- PointSet
- EdgeMesh
- TriMesh
- TetraMesh
- As for simplices, they could be:

```
template <int order,....> Complex{...};
```

- but they are not (at the moment)

Complex (ex: TriMesh)

- A complex is just a collection of simplices

```
template < class VertContainerType, class FaceContainerType >
class TriMesh{
        public:

        /// Set of vertices
        VertContainer vert;
        /// Real number of vertices
        int vn;
        /// Set of faces
        FaceContainer face;
        /// Real number of faces
        int fn;
        ...
        };
```

- Max and min order simplices are the only ones explicitly kept

containers

- Usually vectors
  - Most of the code works also with other generic containers
- Lazy deletion strategy
  - Object that have to be deleted are just marked and purged away later...
    - SetD() IsD() function over simplces
- No edges.
  - Only maximal complexes are usually kept
  - It could be discussed...

Complex

- Topological relations stored optionally inside the simplexes
- Each Simplex knows its own geometric realization

  (a face contains pointers instead of indexes)

Hello Mesh

```
#include <stdio.h>
#include<vcg/simplex/vertex/vertex.h>
#include<vcg/simplex/vertex/with/vn.h>
#include<vcg/simplex/face/with/fn.h>
#include<vcg/simplex/face/with/fcfn.h>
#include<vcg/complex/trimesh/base.h>
#include<vcg/complex/trimesh/create/platonic.h>
#include<vcg/complex/trimesh/update/normal.h>

using namespace vcg;
using namespace std;

class AEdge;   // dummy prototype never used
class AFace;
class AVertex  : public vcg::Vertex< double,AEdge,AFace > {};
class AFace    : public vcg::FaceFCFN< AVertex,AEdge,AFace > {};
class AMesh    : public vcg::tri::TriMesh< vector<AVertex>, vector<AFace> > {};

class CEdge;   // dummy prototype never used
class CFace;
class CVertex  : public vcg::VertexVN< double,CEdge,CFace > {};
class CFace    : public vcg::FaceFN< CVertex,CEdge,CFace > {};
class CMesh    : public vcg::tri::TriMesh< vector<CVertex>, vector<CFace> >{};
```


Hello Mesh

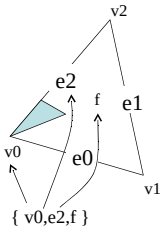
```
int main(int , char **)
{
  AMesh am;
  CMesh cm;
  tri::Tetrahedron(cm);
  tri::Tetrahedron(am);

  printf("Generated mesh has %i vert and %i tri faces\n",cm.vn,cm.fn);

  /// Calculates both vertex and face normals.
  /// The normal of a vertex v is the weigthed average of the normals
  /// of the faces incident on v. Normals are not normalized

  tri::UpdateNormals<CMesh>::PerVertexPerFace(cm);
  printf("Normal of face 0 is %f %f %f",
           cm.face[0].N()[0],cm.face[0].N()[1],cm.face[0].N()[2]);

  tri::UpdateNormals<AMesh>::PerFace(am);

  return 0;
}
```

Surfing a the mesh (I)

- All based on the concept of pos (position)
- a pos is a d-tuple:

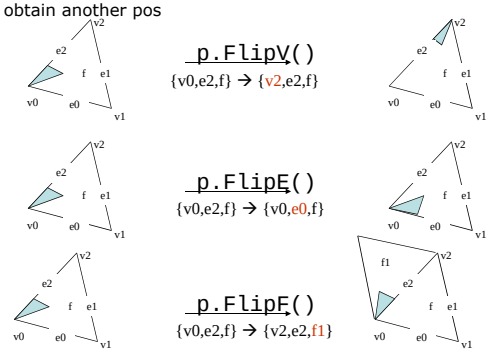
$$p = \{ s_0,\ldots,s_d \}$$

such that each si is a reference to a d-simplex For Meshes is a triple of pointers to a vertex, edge and face



Pos

- any component of a pos can be changed only into another value to obtain another pos

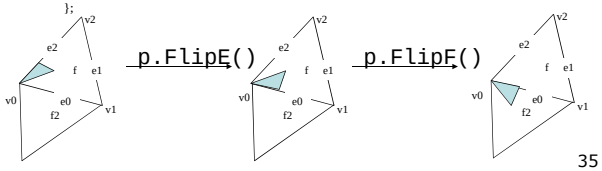


Pos

- Example: running over the faces around a vertex

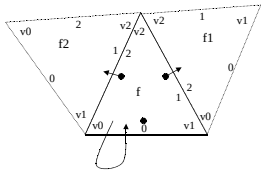
<vcg/simplex/face/pos.h>

```
template <typename FaceType>
class Pos {
    ...
        void NextE()
        {
            assert( f->V(z)==v || f->V((z+1)%3)==v );
            FlipE();
            FlipF();
            assert( f->V(z)==v || f->V((z+1)%3)==v );
        }
    ...
};
```



FF Implementation

- Three pointers to face and three integers in each face:



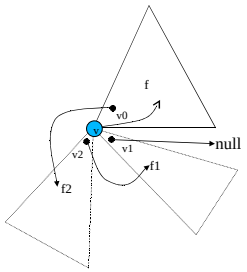

```
f.FFp(1)==&f1
f.FFi(1) == 2

f.FFp(2) == &f2
f.FFi(2) == 1

f.FFp(0) == &f
f.FFi(0) == -1

f.FFp(i)->f.FFp(f.FFi(i)) == &f
```

FF implementation

- Works also for non manifold situations.
  - The pointers in the FF forms a circular list ordered, bidirectional list.
  - It does not hold anymore that `f.FFp(i)->f.FFp(f.FFi(i)) == &f`
  - The pos flip property do not hold any more...

VF Implementation

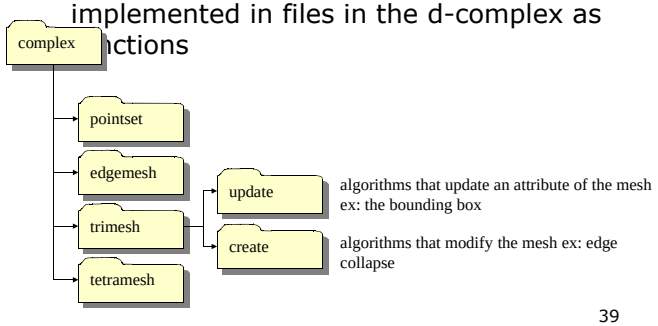
- The list is distributed over the involved faces: No dynamic allocation




```
v.VFp()=&f
v.VFi()=0

f.VFp(0)==&f2
f.VFi(1) == 2

f2.VFp(2) == &f2
f2.VFi(2) == 1

f1.VFp(1) == null
f1.VFi(1) == -1
```


Algorithms

- Algorithms that applies to d-complex are implemented in files in the d-complex as functions



Algorithms (example)

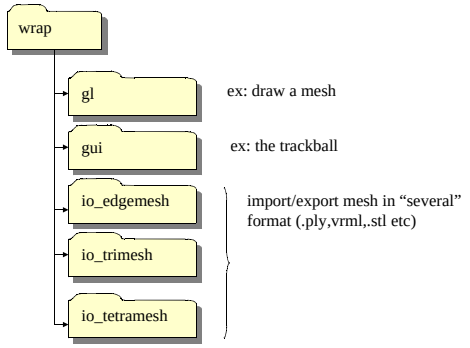
file: vcg/complex/trimesh/update/bounding.h

```
template <class ComputeMeshType>
class UpdateBounding
{
public:
typedef typename MeshType::VertexIterator VertexIterator;

/// Calculates the bounding box (if stored in the current face type)
static void Box(ComputeMeshType &m)
{
  m.bbox.SetNull();
  VertexIterator vi;
  for (vi=m.vert.begin();vi!=m.vert.end();++vi)
        if( !(*vi).IsD() )        m.bbox.Add((*vi).P());
}
}; // end class
```

Usage:

```
MyMeshType m;
....
vcg::tri::UpdateBounding<MyMeshType>::Box(m);
```

Using VCG



ex: draw a mesh

ex: the trackball

import/export mesh in “several” format (.ply,vrml,.stl etc)