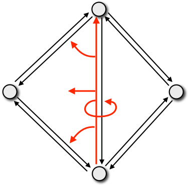
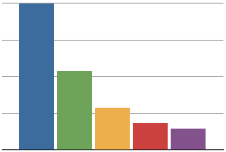
Design, Implementation, and Evaluation of the `Surface_mesh` Data Structure



```cpp
class Vertex_iterator
{
public:

    /// Default constructor
    Vertex_iterator(Vertex v=Vertex()) : hnd_(v) {}

    /// Cast to the vertex the iterator refers to
    operator Vertex() const { return hnd_; }

    /// are two iterators equal?
    bool operator==(const Vertex_iterator& rhs) const
    {
        return (hnd_==rhs.hnd_);
    }

    /// are two iterators different?
    bool operator!=(const Vertex_iterator& rhs) const
    {
        return !operator==(rhs);
    }

    /// pre-increment iterator
    Vertex_iterator& operator++()
    {
        ++hnd_.idx_;
        return *this;
    }

    /// pre-decrement iterator
    Vertex_iterator& operator--()
    {
        --hnd_.idx_;
        return *this;
    }

private:
    Vertex  hnd_;
};
```



Daniel Sieger and Mario Botsch

Graphics & Geometry Processing Group
Bielefeld University
Germany

Universität Bielefeld

CITEC

Outline

Introduction

Design

Implementation

Evaluation
- Memory
- Performance
- Ease of Use

Conclusions

Introduction

Mesh data structures

- Fundamental for research and development
- Time-consuming to implement

Introduction

Mesh data structures

- Fundamental for research and development
- Time-consuming to implement

Existing libraries (Mesquite, CGAL, OpenMesh, ...)

Introduction

Mesh data structures

- Fundamental for research and development
- Time-consuming to implement

Existing libraries (Mesquite, CGAL, OpenMesh, ...)

- Strengths and weaknesses

Introduction

Mesh data structures

- Fundamental for research and development
- Time-consuming to implement

Existing libraries (Mesquite, CGAL, OpenMesh, ...)

- Strengths and weaknesses
- Complexity
- Performance

Introduction

Our approach:

- Systematically analyze **design** choices
- Careful and efficient **implementation**
- Comprehensive **evaluation**

Outline

Introduction

Design

Implementation

Evaluation
- Memory
- Performance
- Ease of Use

Conclusions

Design Goals

Compromise between

- Efficiency
- Memory
- Applicability
- Ease of use

Design Goals

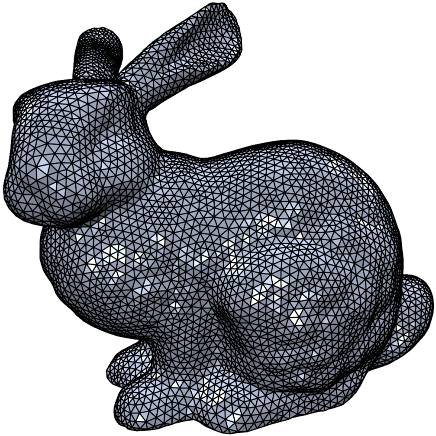
Compromise between

- Efficiency
- Memory
- Applicability
- **Ease of use**

Design Decisions

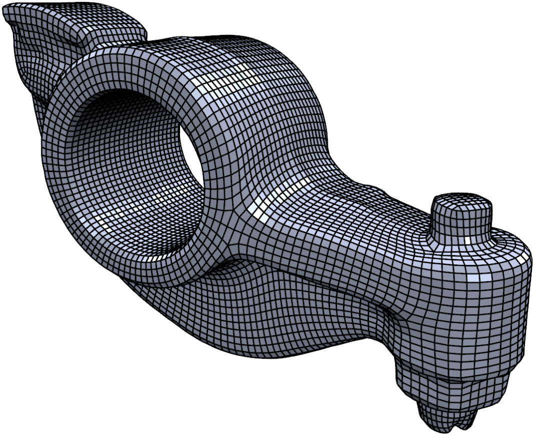
Element Types

Computer Graphics: triangle meshes



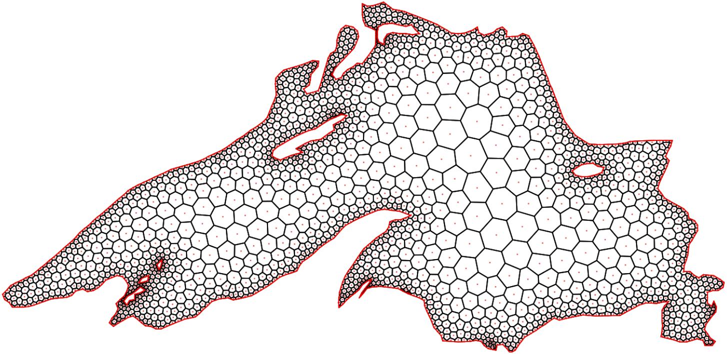
Element Types

Simulation: quad meshes



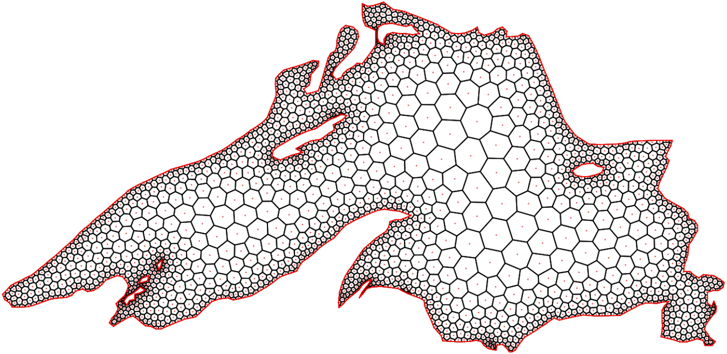
Element Types

Voronoi diagrams, Polygonal Finite Elements



Element Types

Voronoi diagrams, Polygonal Finite Elements



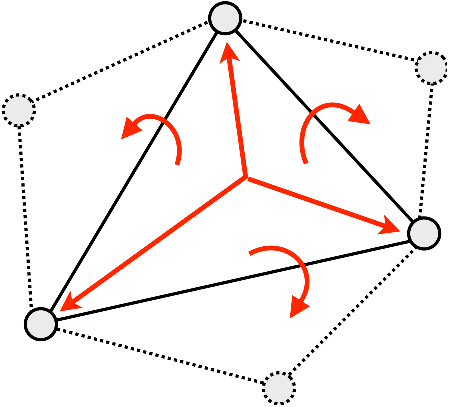
→ support arbitrary polygonal elements

Connectivity Representation

face-based vs. edge-based

Face-Based Representations

- Store faces and references to
  - Defining vertices
  - Neighboring faces



Face-Based Representations

Pros:

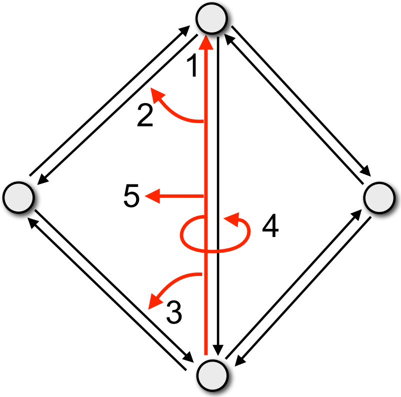
+ Simplicity
+ Memory consumption

Cons:

– Variable size data types
– One-ring traversal
– No explicit edges

Halfedge Data Structure

- Store pairs of oriented halfedges



1. Target vertex
2. Next halfedge
3. Previous halfedge
4. Opposite halfedge
5. Adjacent face

Halfedge Data Structure

Pros:

+ All entities and relations represented
+ Constant size data types
+ Efficient adjacency queries

Cons:

– Memory consumption

Halfedge Data Structure

Pros:

- \+ All entities and relations represented
- \+ Constant size data types
- \+ Efficient adjacency queries

Cons:

- – Memory consumption

→ use a halfedge data structure

Storage Scheme

List-based

+ Easy removal

– Performance
– Memory consumption
– Memory layout

Storage Scheme

List-based

+ Easy removal

– Performance
– Memory consumption
– Memory layout

Array-based

+ Performance
+ Memory consumption
+ Memory layout

– Garbage collection

Storage Scheme

List-based

+ Easy removal

– Performance
– Memory consumption
– Memory layout

Array-based

+ Performance
+ Memory consumption
+ Memory layout

– Garbage collection

⟶ use array-based storage

Entity References

Pointers

- \+ Direct memory access
- – Invalid upon resize
- – Memory consumption

Indices

- \+ Validity checks
- \+ Memory consumption
- – Indirect memory access

Entity References

Pointers

+ Direct memory access

– Invalid upon resize
– Memory consumption

Indices

+ Validity checks
+ Memory consumption

– Indirect memory access

⟶ use indices

Custom Properties

Extended entities

```
class Vertex
{
  Vec3 position;
  Vec3 normal;
  float weight;
};
Vertex  vertices[N];
```

Custom Properties

Extended entities

```
class Vertex
{
  Vec3 position;
  Vec3 normal;
  float weight;
};
Vertex  vertices[N];
```

+ OOP design

– Static at compile-time
– Memory layout

Custom Properties

Extended entities

```
class Vertex
{
  Vec3 position;
  Vec3 normal;
  float weight;
};
Vertex  vertices[N];
```

+ OOP design

– Static at compile-time
– Memory layout

Synchronized arrays

```
Vec3   positions[N];
Vec3   normals[N];
float  weights[N];
```

Custom Properties

Extended entities

```
class Vertex
{
  Vec3 position;
  Vec3 normal;
  float weight;
};
Vertex  vertices[N];
```

+ OOP design

– Static at compile-time
– Memory layout

Synchronized arrays

```
Vec3   positions[N];
Vec3   normals[N];
float  weights[N];
```

+ Dynamic at run-time
+ Memory layout

– Synchronization

Custom Properties

Extended entities

```
class Vertex
{
  Vec3 position;
  Vec3 normal;
  float weight;
};
Vertex  vertices[N];
```

+ OOP design

– Static at compile-time
– Memory layout

Synchronized arrays

```
Vec3  positions[N];
Vec3  normals[N];
float weights[N];
```

+ Dynamic at run-time
+ Memory layout

– Synchronization

⟶ use synchronized arrays

Ease of Use

Genericity

+ Customization

– Accessibility
– Documentation

Simplicity

+ Accessibility
+ Documentation

– Customization

Ease of Use

Genericity

+ Customization

– Accessibility
– Documentation

Simplicity

+ Accessibility
+ Documentation

– Customization

→ strive for simplicity

Summary

Supported element types	→	arbitrary polygons
Connectivity representation	→	halfedges
Storage scheme	→	arrays
Entity references	→	indices
Custom properties	→	synchronized arrays
Ease of use	→	simplicity

Outline

Implementation Overview

- Based on OpenMesh

Implementation Overview

- Based on OpenMesh
- Massively simplified
- Single class, no complicated hierarchy
- Templates reduced to a minimum

Implementation Overview

- Based on OpenMesh
- Massively simplified
- Single class, no complicated hierarchy
- Templates reduced to a minimum
- OpenMesh: 8400 lines of code
- `Surface_mesh` : 2250 lines of code

Example: Smoothing

```
#include <Surface_mesh.h>

int main(int argc, char** argv)
{
    Surface_mesh mesh;

    mesh.read(argv[1]);

    Vertex_property<Point> points =  mesh.get_vertex_property<Point>("v:point");
    Vertex_iterator vit, vend = mesh.vertices_end();
    Vertex_around_vertex_circulator vc, vc_end;

    for (vit = mesh.vertices_begin(); vit != vend; ++vit)
    {
        Point  p(0,0,0);
        Scalar c(0);
        vc = vc_end = mesh.vertices(*vit);
        do
        {
            p += points[*vc];
            ++c;
        }
        while (++vc != vc_end);
        points[*vit] = p / c;
    }

    mesh.write(argv[2]);
}
```

Outline

Introduction

Design

Implementation

Evaluation
- Memory
- Performance
- Ease of Use

Conclusions

Evaluation Overview

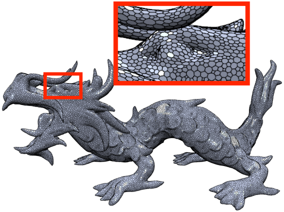
Libraries:

- Mesquite
- CGAL
- OpenMesh

Models

Imp Model
300k vertices, 600k triangles

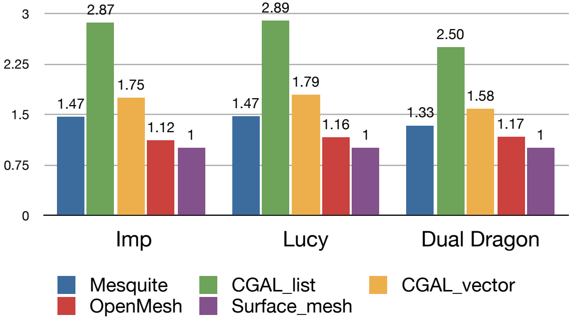
Lucy Model
10M vertices, 20M triangles



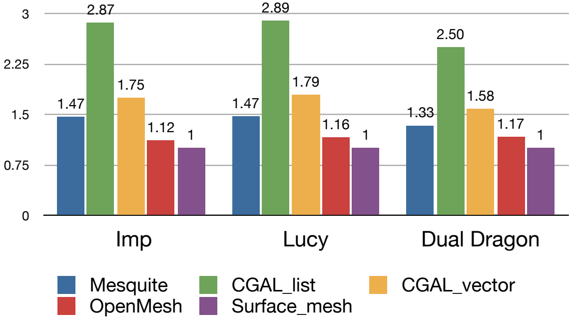
Dual Dragon Model
100k vertices, 50k polygons

Outline

Memory Consumption



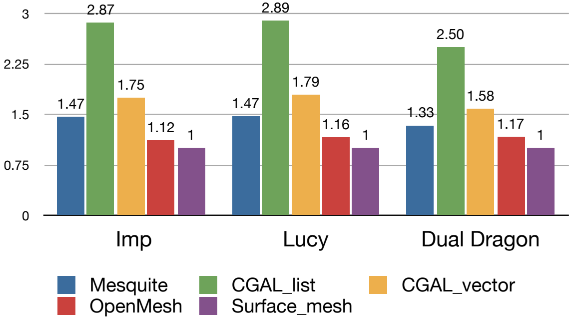
Memory Consumption



Mesquite

- Variable storage
- 64-bit references
- Helper data

Memory Consumption



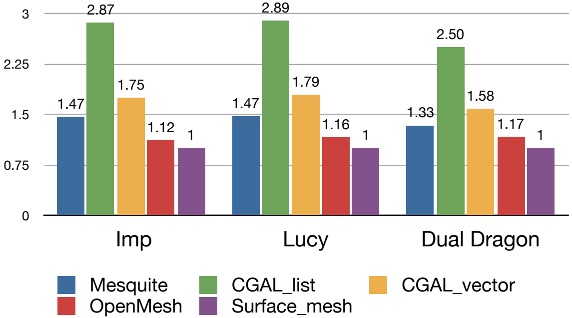
Mesquite

- Variable storage
- 64-bit references
- Helper data

CGAL

- 64-bit references
- List: Generally higher

Memory Consumption



Mesquite

- Variable storage
- 64-bit references
- Helper data

CGAL

- 64-bit references
- List: Generally higher

OpenMesh

- Status information

Outline

Introduction

Design

Implementation

Evaluation

- Memory
- Performance
- Ease of Use

Conclusions

Benchmarks Overview

Circulator Faces around vertex, vertices around face

Benchmarks Overview

Circulator	Faces around vertex, vertices around face
Barycenter	Compute barycenter of vertices

Benchmarks Overview

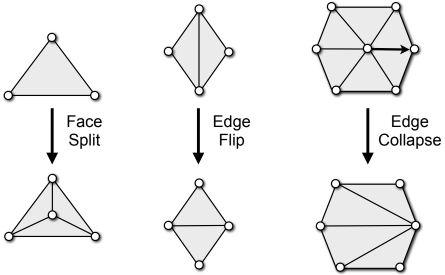
Circulator	Faces around vertex, vertices around face
Barycenter	Compute barycenter of vertices
Normals	Compute and store face & vertex normals

Benchmarks Overview

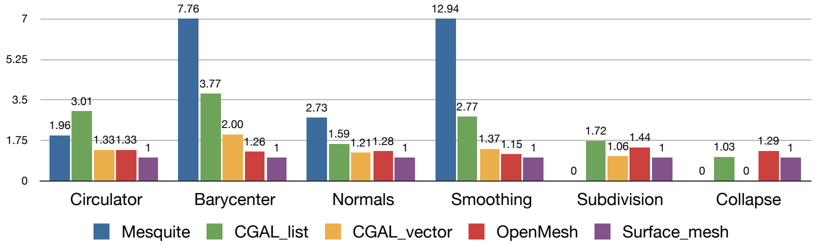
Circulator	Faces around vertex, vertices around face
Barycenter	Compute barycenter of vertices
Normals	Compute and store face & vertex normals
Smoothing	Move vertices to barycenter of neighbors

Benchmarks Overview

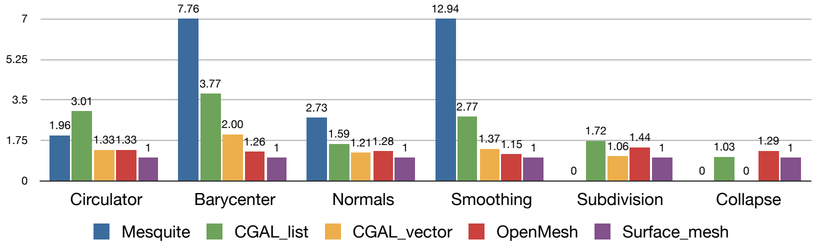
Circulator	Faces around vertex, vertices around face
Barycenter	Compute barycenter of vertices
Normals	Compute and store face & vertex normals
Smoothing	Move vertices to barycenter of neighbors
Subdivision	Face split & edge flip
Collapse	Face split & edge collapse



Results: Imp Model



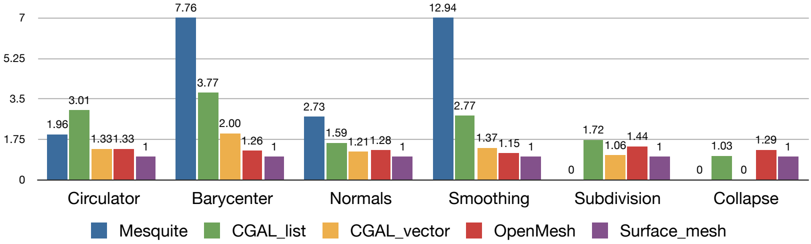
Results: Imp Model



Mesquite

- Virtual functions
- One-ring traversal
- Variable storage
- No topology changes

Results: Imp Model



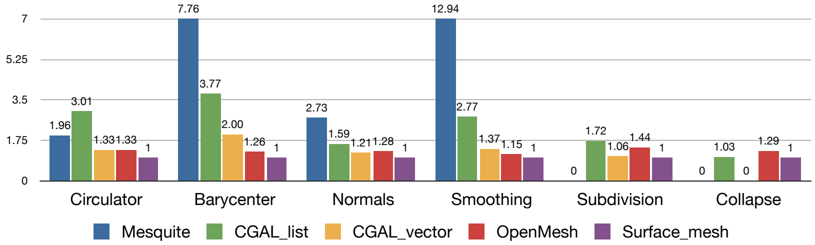
Mesquite

- Virtual functions
- One-ring traversal
- Variable storage
- No topology changes

CGAL

- Extended entities
- List: Memory layout
- Vector: No removal

Results: Imp Model



Mesquite

- Virtual functions
- One-ring traversal
- Variable storage
- No topology changes

CGAL

- Extended entities
- List: Memory layout
- Vector: No removal

OpenMesh

- Circulators
- Properties

Outline

Simplicity & Properties

Working with a custom edge property in `Surface_mesh`

```
Surface_mesh mesh;

// allocate property storing a point per edge
Edge_property<Point> edge_points
  = mesh.add_edge_property<Point>("property-name");

// access the edge property like an array
Edge e;
edge_points[e] = Point(x,y,z);

// remove property and free memory
mesh.remove_edge_property(edge_points);
```

Simplicity & Properties

Just declaring a custom edge property in CGAL

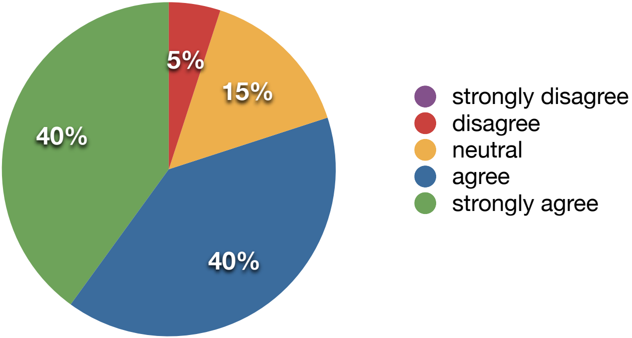
```
typedef CGAL::Simple_cartesian<double>  Kernel;
typedef Kernel::Point_3                 Point_3;

template <class Refs> struct My_halfedge
  : public CGAL::HalfedgeDS_halfedge_base<Refs>
{
    Point_3 halfedge_point;
};

class Items : public CGAL::Polyhedron_items_3
{
public:
    template <class Refs, class Traits>
    struct Halfedge_wrapper
    {
        typedef My_halfedge<Refs> Halfedge;
    };
};

typedef CGAL::Polyhedron_3<Kernel, Items>  Mesh;
```

User Feedback

- SGP 2011 graduate course on geometric modeling
- 18 participants, varying experience
- Q: Is `Surface_mesh` easy to use and understand?

User Feedback

- SGP 2011 graduate course on geometric modeling
- 18 participants, varying experience
- Q: Is `Surface_mesh` easy to use and understand?



User Feedback: Comments

“It is easy to use for users that have experience with CGAL or OpenMesh. It is nice to have a template-free structure.”

“Much easier than OpenMesh.”

“I thought the data structure was quite easy to understand even though I have never worked with a real mesh data structure before.”

Outline

Introduction

Design

Implementation

Evaluation

- Memory
- Performance
- Ease of Use

Conclusions

Conclusions

Our approach:

- Systematically analyze **design** choices
- Careful and efficient **implementation**
- Comprehensive **evaluation**

Conclusions

Our approach:

- Systematically analyze **design** choices
- Careful and efficient **implementation**
- Comprehensive **evaluation**

Our implementation of `Surface_mesh` :

- Easy to use
- Highly efficient
- Low memory consumption

Conclusions

Our approach:

- Systematically analyze **design** choices
- Careful and efficient **implementation**
- Comprehensive **evaluation**

Our implementation of `Surface_mesh` :

- Easy to use
- Highly efficient
- Low memory consumption

Other libraries:

- Much more functionality beyond a pure data structure

Limitations & Future Work

Limitations:

- Single `Surface_mesh` declaration in one application
- Two-manifold surface meshes only

Limitations & Future Work

Limitations:

- Single `Surface_mesh` declaration in one application
- Two-manifold surface meshes only

Future work:

- Volumetric meshes

Limitations & Future Work

Limitations:

- Single `Surface_mesh` declaration in one application
- Two-manifold surface meshes only

Future work:

- Volumetric meshes
- Include more libraries (VCGLib, LR, yours?)
- Establish standard set of benchmarks

→ `http://graphics.uni-bielefeld.de`

Thanks

... for your attention.