Progressive Meshes (96)

Hugues Hoppe

and

Efficient Implementation of P-Meshes (98)

Hugues Hoppe

TongKe Xue

Problems with Large Meshes

- Expensive to store
- Expensive to transmit
- Exprensive to renders
-
- --> Motivates new problems

Motivated Problems

- Mesh simplification
- Level-of-detail approximation
  - smooth geomorphs
- Progressive transmission
- Mesh compression
- Selective refinement

'96 paper claims ...

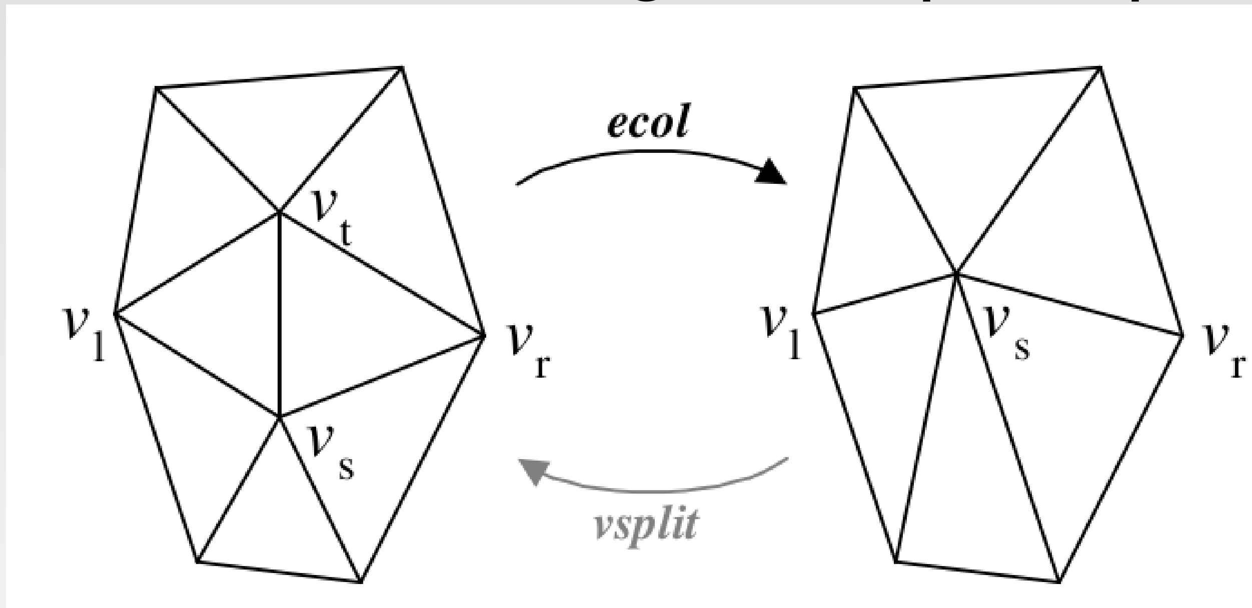
- Novel progressive mesh representation; progressive transmission; concise encoding:
  - probably true
- Selective refinement
  - questionable; requires an (elegant) hack not in the paper
- Novel simplification procedure
  - seems very hackey (in the bad sense)

Table of Contents

- General Idea + What P-Meshes provide
- How to Store Efficiently / Compression
- How to Construct P-Meshes

Idea 1 (Limitation 1)

- Limit ourselves to: edge collapse operations



- New vertex is either old, new, or avg(old, new)

Idea 2 (Iterate!)

- M = original model; construct a series of models M=M_0, M_1, M_2, ... M_n=M* where from M_k to M_{k+1}, we kill a single edge
- M* = coarse model; store the diffs between M_k as M_{k+1} as vertex splits

Important Detail

- A vertex split (inverse of edge collapse) consists of:
  - (s, l, r, t, A)
- We add a vertex *t* near an (existing) vertex *s*, adding faces *stl* and *tsr*, and update some attributes (color, texture, shader, ...) with A
- vertex split == diff

Free Features 1

- Progressive Transmission
  - Send M* ... then send the diffs later
- Smooth Geomorphing:
  - We're just adding in verticies; easy to linearly interpolate diffs
    - Draw on white board
- Mesh Compression
  - discussed later

Selective Refinement

- We don't have to apply the diff's in order
- for a (s, l, r, t, A) vsplit, if l, r, and s are already there ... we can use it
- if *l* or *r* are not there, but their 'closest-living-ancestor' are adjacent to *s*, then we can pseudo-apply
- ugly detail not mentioned in paper: we can still guarantee same mesh in the end

Movies!

Table of Contents

- General Idea + What P-Meshes provide
- How to Store Efficiently / Compression
- How to Construct P-Meshes

Faces, Wedges, Corners

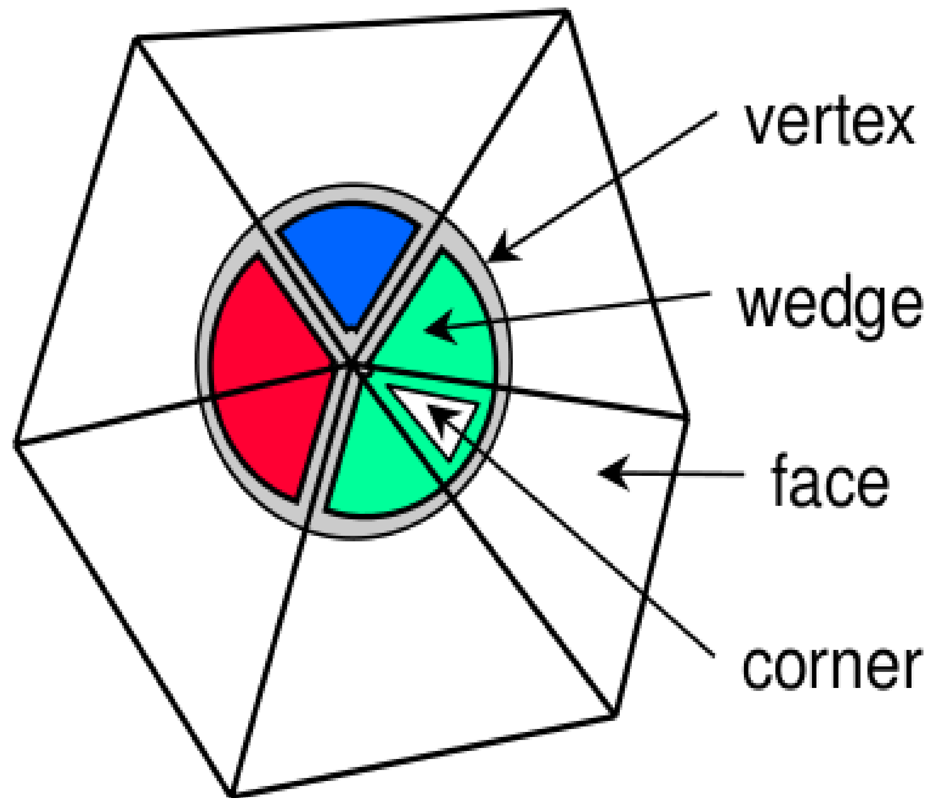


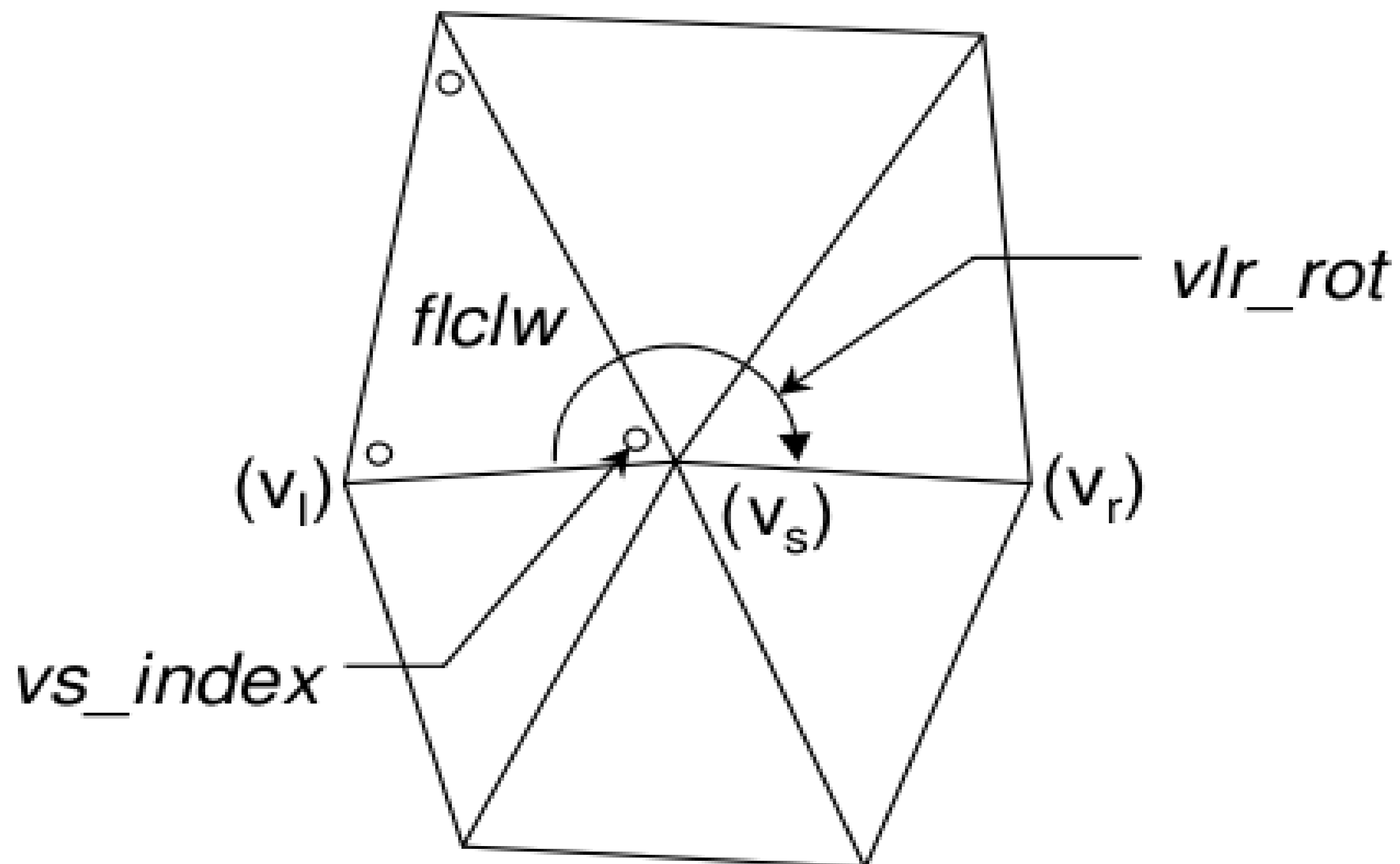
Figure 3: Illustration of vertices, wedges, and faces. In this example, the central vertex has 6 adjacent corners which are partitioned into 3 wedges.

vsplit structure

```
struct Vsplit {
    int flclw;                          // a face in neighborhood of vsplit
    short vlr_rot;                      // encoding of vertex v_r
    struct {
        short vs_index : 2;             // index (0..2) of v_s within flclw
        short corners : 10;             // corner continuities in Figure 9
        short ii : 2;                   // geometry prediction of Figure 10
        short matid_predict : 2;        // are fl_matid,fr_matid required?
    } code;                             // set of 4 bit-fields (16-bit total)
    short fl_matid;                     // matid of face fl if not predicted
    short fr_matid;                     // matid of face fr if not predicted
    VertexAttribD vad_l, vad_s;
    Array<WedgeAttribD> wads;
};
```

efficient vertex split

Figure 6: Vertex split data structure.



corners

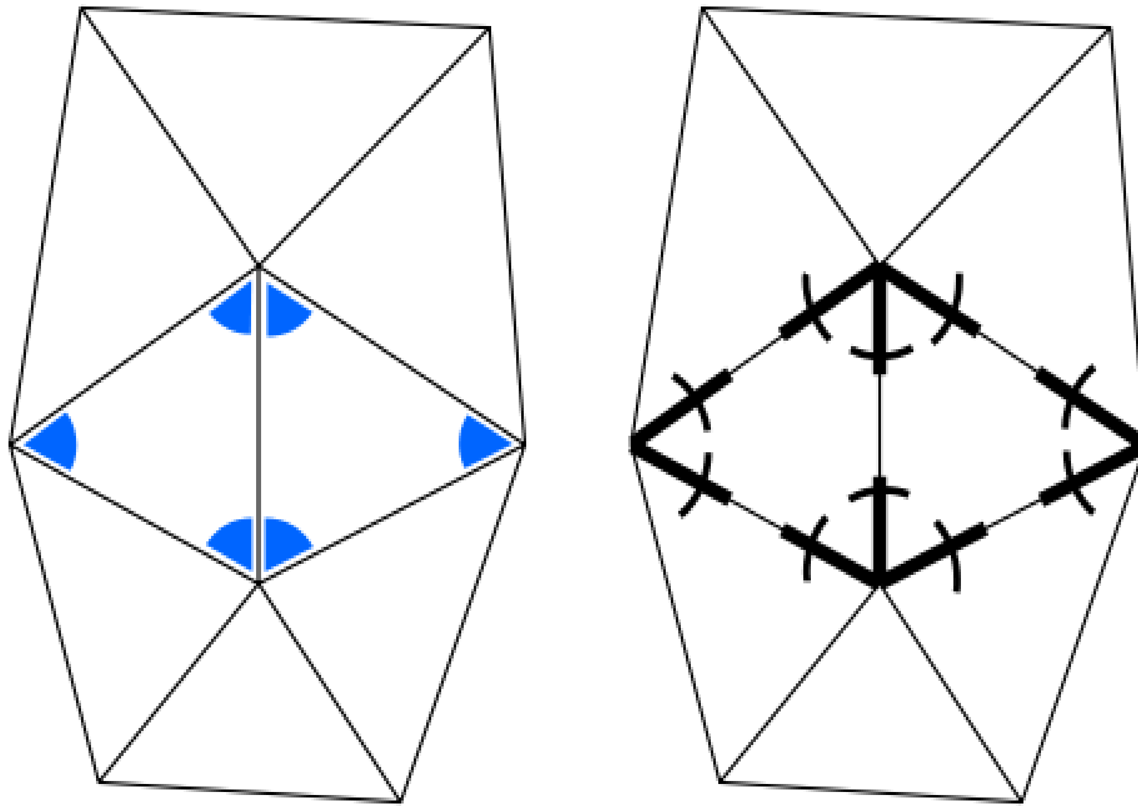


Figure 9: The 6 new corners introduced by a vertex split, and the 10-bit field *corners* used to record the continuity of corner attributes.

ii

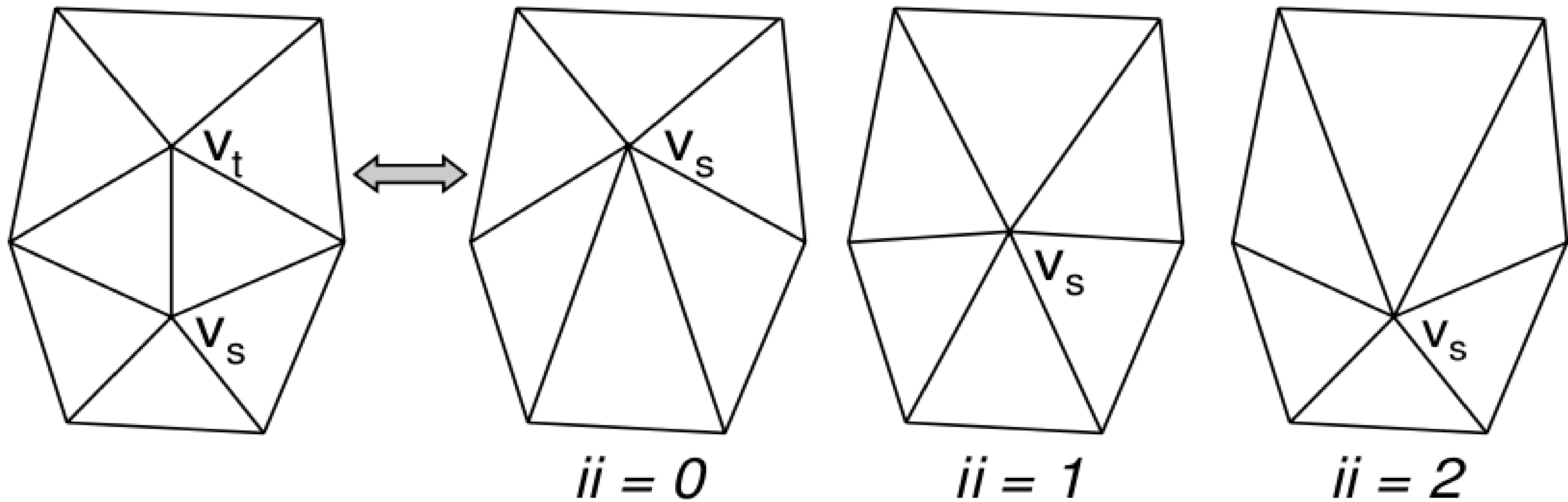


Figure 10: The Vsplit parameter *ii* used for geometry prediction.

'Standard' Mesh

```
struct VertexAttrib {
    Point point;
};
struct WedgeAttrib {
    Vector normal;
    UV uv;
};

struct Vertex {
    VertexAttrib attrib;
};
struct Wedge {
    int vertex;
    WedgeAttrib attrib;
};
```

```
struct Face {
    int wedges[3];          // wedges at corners of the face
    int fnei[3];            // 3 face neighbors
    short matid;            // material identifier
};

struct Mesh {
    Array<Vertex> vertices;
    Array<Wedge> wedges;
    Array<Face> faces;
    Array<Material> materials;
};
```

PMesh

```
struct PMesh {
    Mesh base_mesh;                      // base mesh M^0
    Array<Vsplit> vsplits;               // {vsplit_0, . . . , vsplit_{n-1}}
    int full_nvertices;                  // number of vertices in M^n
    int full_nwedges;                    // number of wedges in M^n
    int full_nfaces;                     // number of faces in M^n
};
```

M vs M_0

Model	Original mesh M^n			Base mesh M^0			n
	#vertices	#wedges	#faces	#vertices	#wedges	#faces	
garethman	801	1,207	1,586	31	84	46	770
cessna	6,795	9,533	13,546	46	75	48	6,749
bigship	8,536	8,847	17,068	24	59	44	8,512
dunebuggy	11,322	11,674	22,444	513	568	826	10,809
gameguy	21,412	25,095	42,712	31	50	27	21,381
drumset	34,794	59,834	68,776	963	2,192	1,114	33,831
chandelier	36,627	55,289	72,346	2,140	4,930	3,372	34,487
bunny	34,835	34,835	69,473	13	13	18	34,822
dragon	429,753	429,753	859,586	259	259	598	429,494
buddha	517,924	517,924	1,036,260	942	942	2,296	516,982
gcanyon	360,000	360,000	717,602	3	3	1	359,997

Table 1: Statistics for the various data sets.

iteration / space requirements

Model	Iteration rates (verts/sec)		Space for M^n (bits/vertex)					
	goto(M^n)	goto(M^0)	Mesh			PMesh		
			memory	gzip	arith.	memory	gzip	arith.
garethman	n/a	n/a	607	257	214	541	221	111
cessna	105,000	149,000	589	232	227	517	152	86
bigship	112,000	158,000	519	241	199	455	189	105
dunebuggy	97,000	135,000	516	230	208	461	168	88
gameguy	92,000	126,000	544	240	223	477	158	80
drumset	79,000	108,000	648	272	276	572	179	100
chandelier	81,000	112,000	607	249	257	542	170	98
bunny	80,000	107,000	511	247	209	448	148	74
dragon	76,000	101,000	512	248	235	448	132	64
buddha	75,000	100,000	512	248	237	449	132	65
gcanyon	71,000	94,000	511	223	233	448	97	58

Table 2: PM iteration rates and space requirements.

vsplit fields (slightly lossy)

Model	Avg. $|wad|$	*flclw*	*vs_index*	*vlr_rot*	*corners+ii+ matid_pred*	*fl_matid+ fr_matid*	VertexAttribD		WedgeAttribD		Σ
							vad_l	*vad_s*	Δ*normal*	Δ*uv*	
garethman	1.49	8.4	1.6	1.6	4.6	0.1	36.2	21.7	31.4	0.0	105.5
cessna	1.42	11.3	1.6	1.9	4.1	0.1	29.1	12.5	24.3	0.0	85.0
bigship	1.02	11.6	1.6	2.0	1.2	0.0	30.2	18.0	18.3	22.2	105.1
dunebuggy	1.03	12.2	1.6	2.0	0.6	0.0	27.5	20.0	20.6	0.0	84.5
gameguy	1.18	13.0	1.6	1.7	2.5	0.0	26.3	13.7	21.2	0.0	80.1
drumset	1.74	13.8	1.6	2.2	4.6	0.7	25.3	15.6	32.0	0.0	95.8
chandelier	1.46	14.0	1.6	2.1	1.9	0.0	23.9	15.1	28.6	0.0	87.3
bunny	1.00	13.6	1.6	1.4	0.1	0.0	28.2	15.3	13.7	0.0	74.0
dragon	1.00	17.3	1.6	2.0	0.0	0.0	21.6	8.9	12.9	0.0	64.2
buddha	1.00	17.6	1.6	2.0	0.0	0.0	21.1	8.4	13.7	0.0	64.4
gcanyon	1.00	17.0	1.6	1.7	0.1	0.0	21.5	6.4	9.7	0.0	58.1

Table 3: Space of Vsplit fields (bits/*vsplit*), with arithmetic coding and variable-length delta encoding.

approximate/restrict more

Model	Avg. $\|wad\|$	$\Delta flclw$	vs_index	vlr_rot	$corners+ii+$ $matid_pred$	fl_matid+ fr_matid	VertexAttribD		WedgeAttribD		Σ
							vad_l	vad_s	$\Delta normal$	Δuv	
garethman	1.49	6.1	1.6	1.6	4.6	0.1	36.2	0.0	0.0	0.0	50.1
cessna	1.42	6.8	1.6	1.9	4.1	0.1	29.2	0.0	0.0	0.0	43.7
bigship	1.02	6.5	1.6	2.0	1.2	0.0	30.2	0.0	0.0	22.2	63.6
dunebuggy	1.03	6.8	1.6	2.0	0.6	0.0	27.3	0.0	0.0	0.0	38.3
gameguy	1.18	6.8	1.6	1.7	2.5	0.0	26.3	0.0	0.0	0.0	39.0
drumset	1.74	6.8	1.6	2.2	4.6	0.7	25.2	0.0	0.0	0.0	41.1
chandelier	1.46	6.9	1.6	2.1	1.9	0.0	23.9	0.0	0.0	0.0	36.5
bunny	1.00	6.3	1.6	1.4	0.1	0.0	28.2	0.0	0.0	0.0	37.7
dragon	1.00	6.7	1.6	2.0	0.0	0.0	21.5	0.0	0.0	0.0	31.8
buddha	1.00	6.8	1.6	2.0	0.0	0.0	21.1	0.0	0.0	0.0	31.5
gcanyon	1.00	6.6	1.6	1.7	0.1	0.0	21.4	0.0	0.0	0.0	31.5

Table of Contents

- General Idea + What PM's provide
- How to Store Efficiently / Compression
- How to Construct P-meshes
  - This can be done better.

'related' background

- from his 93 paper
- $$E(M) = E_{dist}(M) + E_{rep}(M) + E_{spring}(M)$$
- $$E_{dist}(M) = \sum_{i} d^2(\mathbf{x}_i, \phi_V(|K|))$$
- $$E_{spring}(M) = \sum_{\{j,k\} \in K} \kappa \|\mathbf{v}_j - \mathbf{v}_k\|^2$$
-
- simulated annealing / energy like

actual energy function

- $E(M) = E_{dist}(M) + E_{spring}(M) + E_{scalar}(M) + E_{disc}(M)$
- 'scalar' to minimize scalar changes; disc to ensure discontinuities aren't eliminated
- throw all edges on a priority queue + pop

Object	Original $\hat{M}$		Base M^0		User param.		$\|X_{disc}\|$	V	Time
	m_0+n	#faces	m_0	#faces	$\|X\|-(m_0+n)$	c_{color}		$\frac{\text{bits}}{n}$	mins
cessna	6,795	13,546	97	150	100,000	-	46,811	46	23
terrain	33,847	66,960	3	1	0	-	3,796	46	16
mandrill	40,000	79,202	3	1	0	0.1	4,776	31	19
radiosity	78,923	150,983	1,192	1,191	200,000	0.01	74,316	37	106
fandisk	6,475	12,946	27	50	10,000	-	5,924	50	19

E_{dist} + E_{spring}

- Claims can't be evaluated efficiently.
- $$E_{dist}(M) = \sum_i d^2(\mathbf{x}_i, \phi_V(|K|))$$
- $$E_{spring}(M) = \sum_{\{j,k\} \in K} \kappa \|\mathbf{v}_j - \mathbf{v}_k\|^2$$
- Possible leftover from mesh optimization (93):

Inner loop: For each candidate mesh transformation, the algorithm computes $E_{K'} = \min_V E_{dist}(V) + E_{spring}(V)$ by optimizing over the vertex positions V. For the sake of efficiency, the algo-

E_{scalar}

- optimizes only for scalar values at vertices / corners $$E_{scalar}(\underline{V}) = (c_{scalar})^2 \sum_i \|\underline{\mathbf{x}}_i - \phi_{\underline{V}}(\mathbf{b}_i)\|^2$$
- claims this works for (r,g,b) values after clipping
- for normals: just interpolates

E_{disc}

- discontinuity curves defined by material boundaries / large changes in corner scalar attributes
- sample additional set of points on discontinuity curve; penalize for distance away from curve

E_{disc} example

Discussion