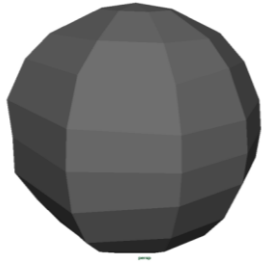
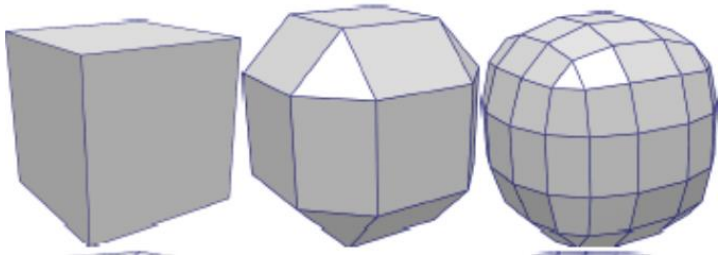
PREDSTAVITVE PREDMETOV

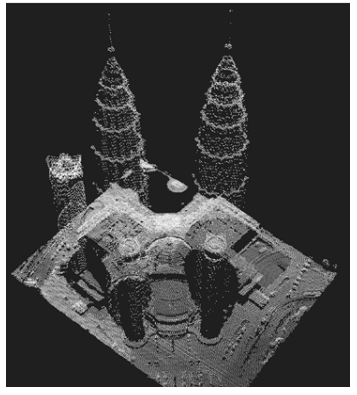
Kako predstavimo 3D predmete v RG?

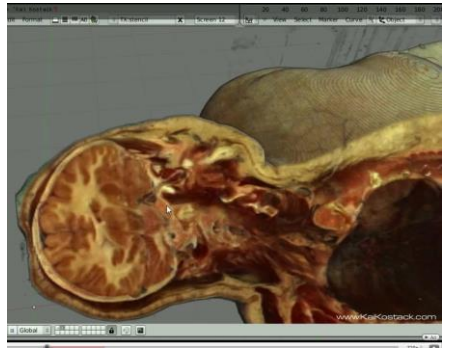
- Ploskve
  - **poligonske mreže**
  - **deljene ploskve**
  - **parametrične**
  - implicitne

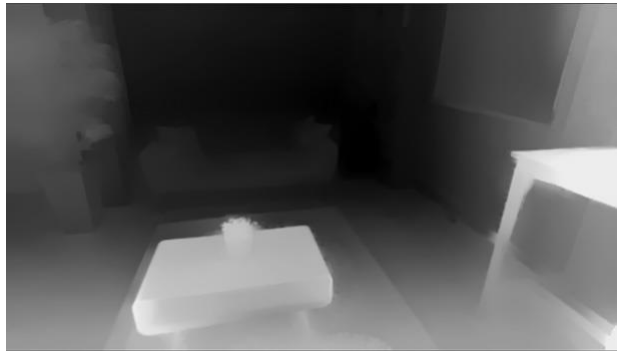




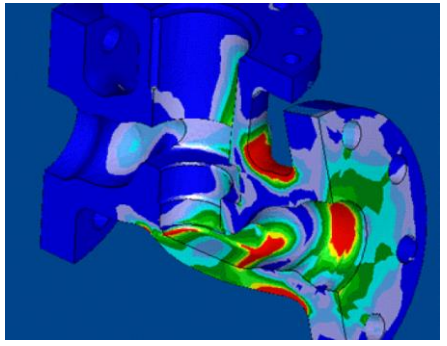
- Neposredno z zajetimi podatki
  - voksli
  - oblaki točk
  - globinske slike

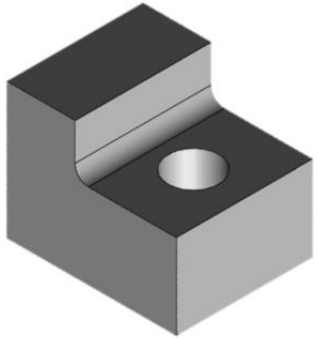


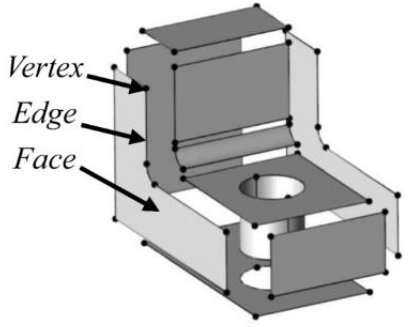




- Polna telesa
  - osmiška drevesa
  - CSG (constructive solid geometry)
  - B-rep (boundary representations)



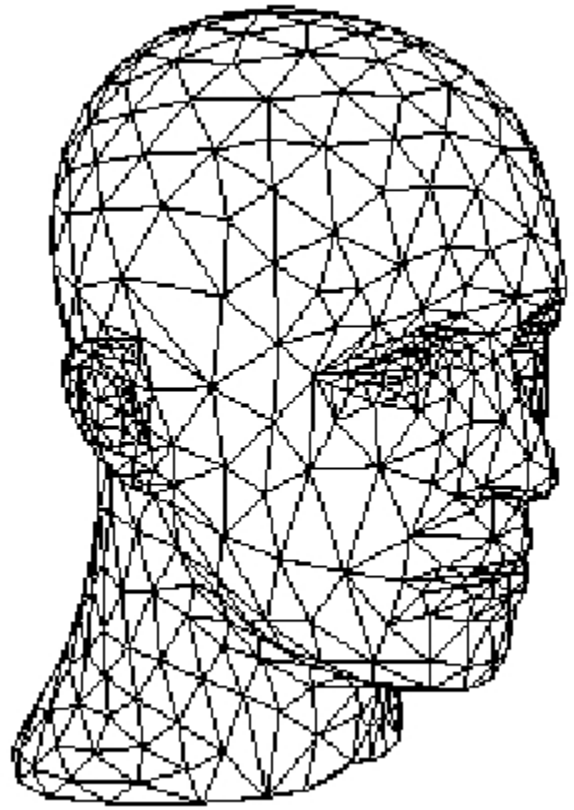




Zakaj različne predstavitve

Želimo predstavitev, ki bo čimbolj:

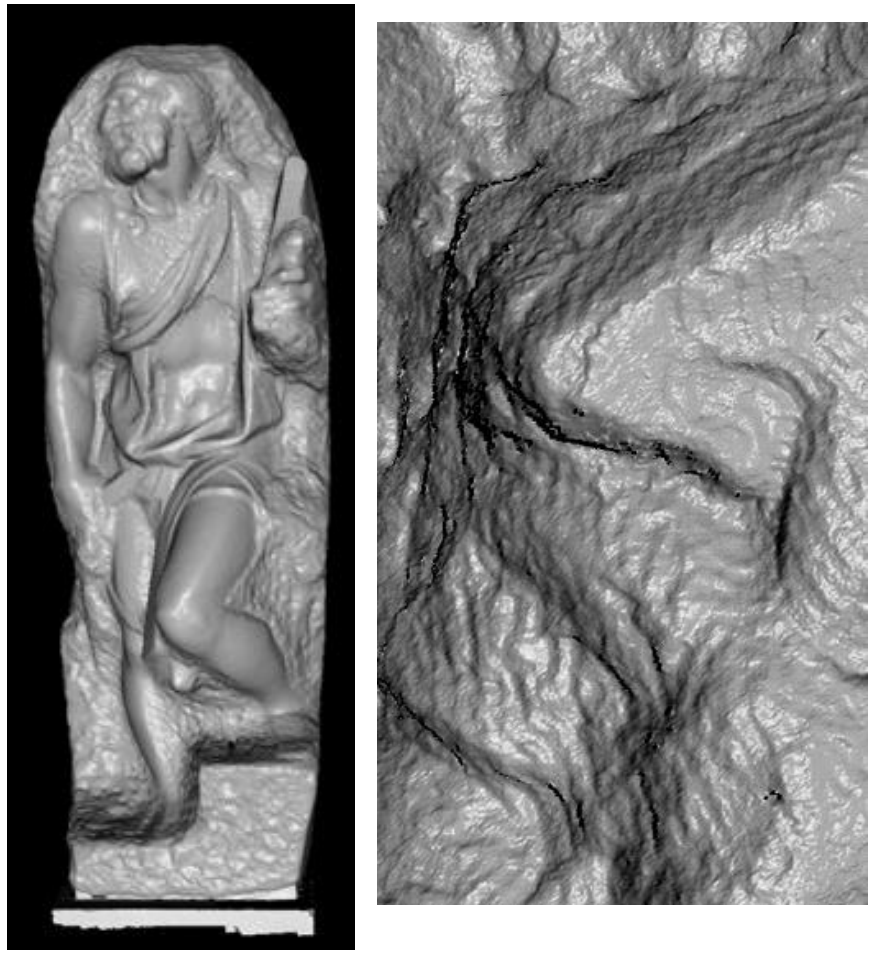
- natančna
- zgoščena
- intuitivna
- podpirala poljubno topologijo
- zvezna oz. gladka
- učinkovita za prikaz
- učinkovita za operacije kot je računanje presekov



Poligonske mreže in povpraševanja

Poligoni

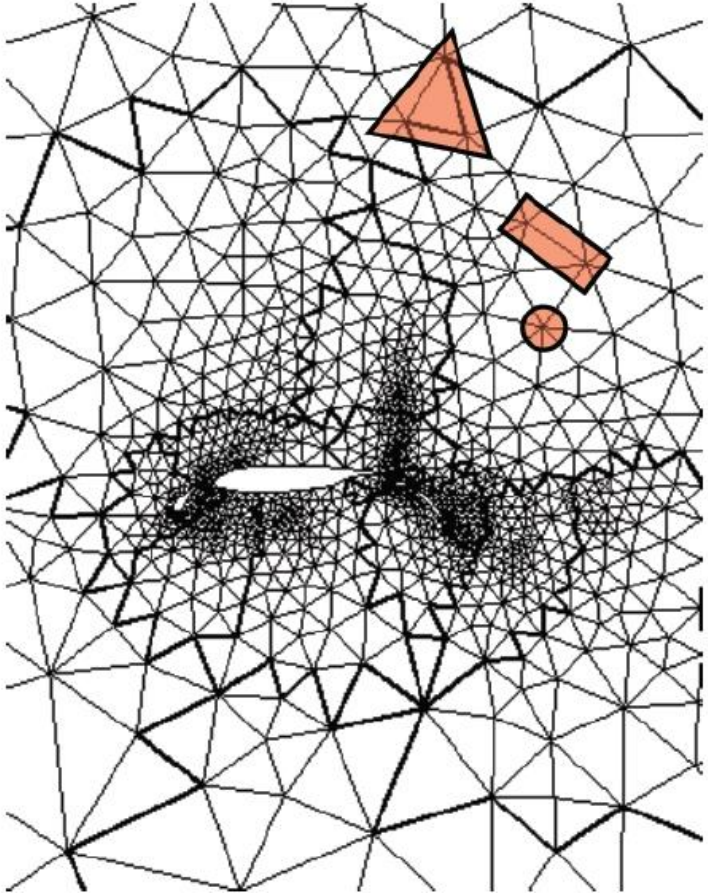
- 3D predmet predstavimo z množico poligonov
  - navadno **trikotnikov** (planarni, konveksni)
- Vprašanje: **kako predstaviti** množico poligonov? Želimo:
  - učinkovita povpraševanja (npr. kateri so sosednji poligoni)
  - učinkovito porabo pomnilnika



St. Matthew
(>370 milijonov trikotnikov)
The Digital Michelangelo Project

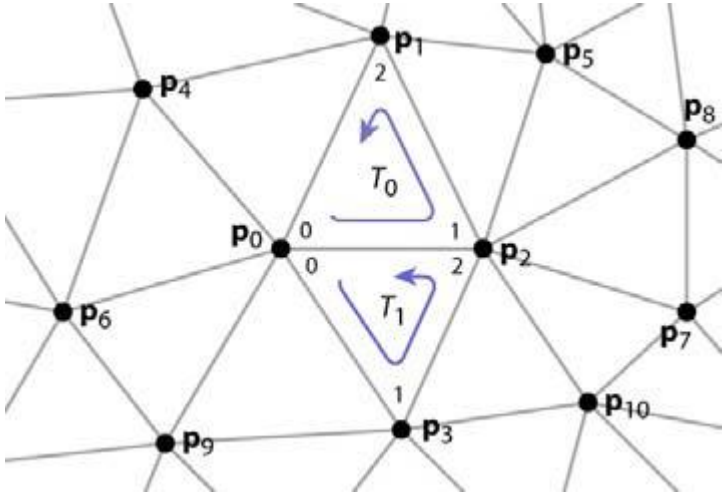
Povpraševanja

- **Povpraševanja** so sestavni del operacij pri izrisovanju, interakciji in spreminjanju 3D modelov
- Vprašanja, na katere pogosto odgovarjamo so iskanje sosedov, npr. *najdi*:
  - oglišča poligona (V-F)
  - oglišča roba (V-E)
  - ploskve okoli oglišča (F-V)
  - ploskve okoli roba (F-E)
  - robove okoli ploskve (E-F)
  - robove oglišča (E-V)
  - ploskve okoli ploskve (F-F)
  - …



Indeksirani poligoni

- Privzeto v OpenGL/WebGPU poligone definiramo z **indeksi oglišč**
  - posebej imamo shranjena oglišča, posebej poligone
- Topologija (povezanost) in geometrija (koordinate oglišč) sta **ločeni**
- Povprečna poraba prostora?
  - v povprečju
    - št. poligonov (#F) ~ 2*št. oglišč (#V)
    - št. robov (#E) ~ 3*št. oglišč (#V)
  - (3+2*3) * #V = 9 * #V
- Omejena povpraševanja
  - le V-F, V-E, E-F so učinkovita

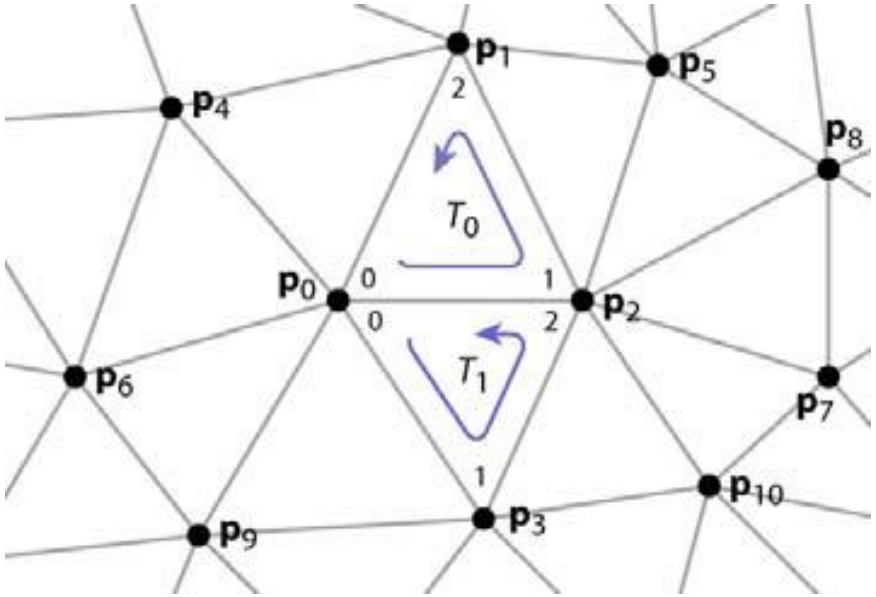


verts[0]	x_0, y_0, z_0
verts[1]	x_1, y_1, z_1
	x_2, y_2, z_2
	x_3, y_3, z_3
	⋮

tInd[0]	0, 2, 1
tInd[1]	0, 3, 2
	⋮

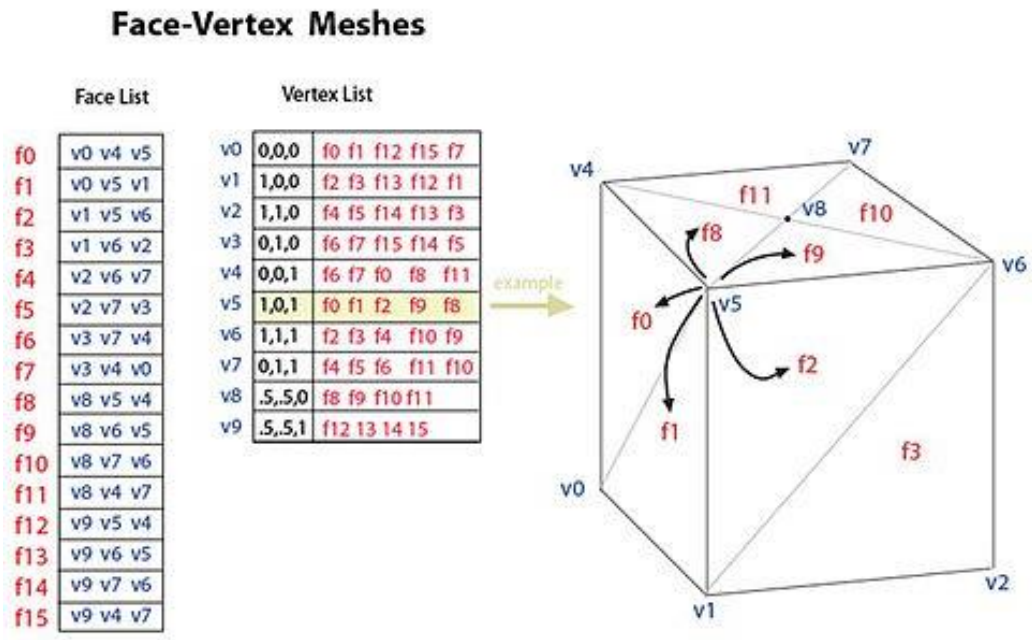
Kako bolje predstaviti topološke informacije

- Lahko bi imeli
  - seznam poligonov z robovi in oglišči
  - seznam oglišč z robovi in poligoni
  - seznam robov z oglišči in poligoni
- Potratno!
  - iščemo bolj učinkovite predstavitve



Predstavitev ploskev-oglišče

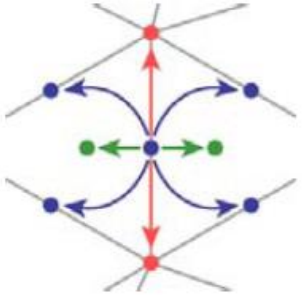
- *Face-vertex mesh*
- Vsakemu oglišču dodamo še seznam ploskev, katerih del je
- Povpraševanja:
  - V-F, V-E, E-F že takoalitako
  - Poenostavi se F-V (imamo tabelo)
  - E-V lahko dobimo s sprehodom po sosednjih ploskvah (preko F-V)
- Poraba prostora:
  - cca (9 + ~4) * *#V*



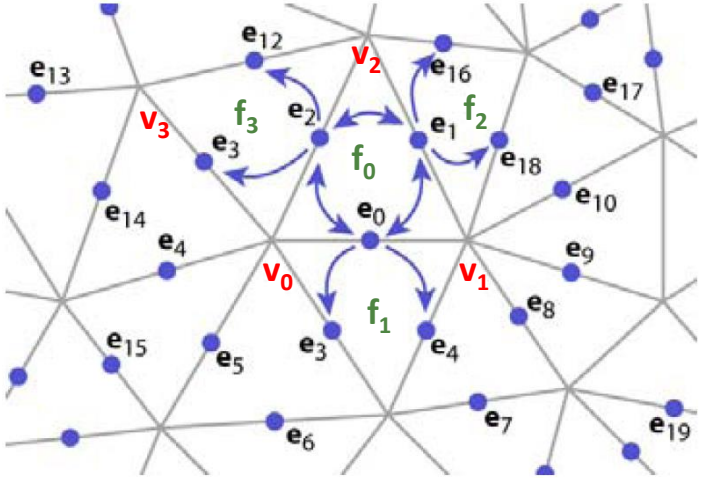
Vir: *wiki*

Winged-edge predstavitev

- Uporablja **robove** za določanje sosednosti
- Vsak **usmerjen rob** kaže na:
  - levi in desni prednji rob
  - levi in desni zadnji rob
  - prednje in zadnje oglišče
  - levi in desni poligon



- Vsak **poligon in oglišče** kažeta na en rob
- Enostavno iskanje vseh osnovnih povpraševanj
  - npr. F-V: V->E0,E1,E2…->F0,F1,F2…
- Porabi več prostora, vendar je bolj splošna
  - (9 + 3*8 + 1 + 2) * #V = (9+27) * #V



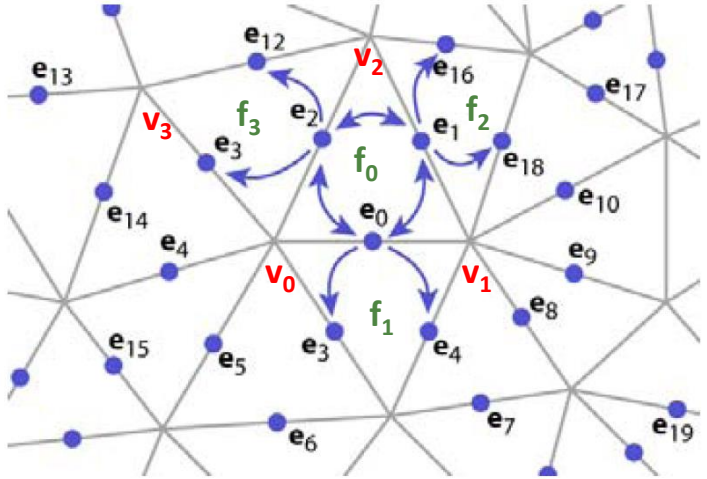
e_0	v_0	v_1	e_1	e_4	e_2	e_3	f_0	f_1
e_1	v_1	v_2	e_2	e_{16}	e_0	e_{18}	f_0	f_2
e_2	v_2	v_0	e_0	e_3	e_1	e_{12}	f_0	f_1
…								

f_0	e_0
f_1	e_3
f_2	e_1

v_0	e_0
v_1	e_1
v_2	e_2

Winged-edge predstavitev

- Primer: želimo najti vse sosede ploskve f_0
  - najdemo rob e_0, ki pripada ploskvi f_0
    - preberemo iz tabele
  - iz roba e_0 se sprehodimo po vseh robovih ploskve e_i preko kazalcev na sosednje robove
    - v isti smeri
  - za vsak $\boldsymbol{e_i}$ dobimo sosednjo ploskev f_i
    - preberemo iz tabele



e_0	v_0	v_1	e_1	e_4	e_2	e_3	f_0	f_1
e_1	v_1	v_2	e_2	e_{16}	e_0	e_{18}	f_0	f_2
e_2	v_2	v_0	e_0	e_3	e_1	e_{12}	f_0	f_1
…								

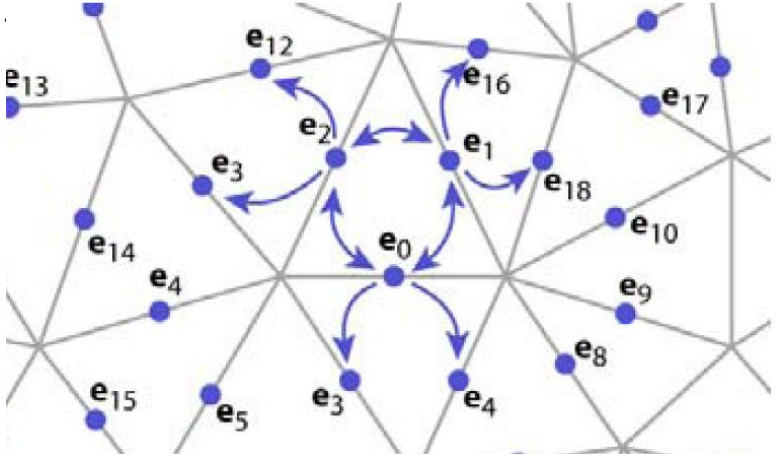
f_0	e_0
f_1	e_3
f_2	e_1

v_0	e_0
v_1	e_1
v_2	e_2

Winged-edge primer

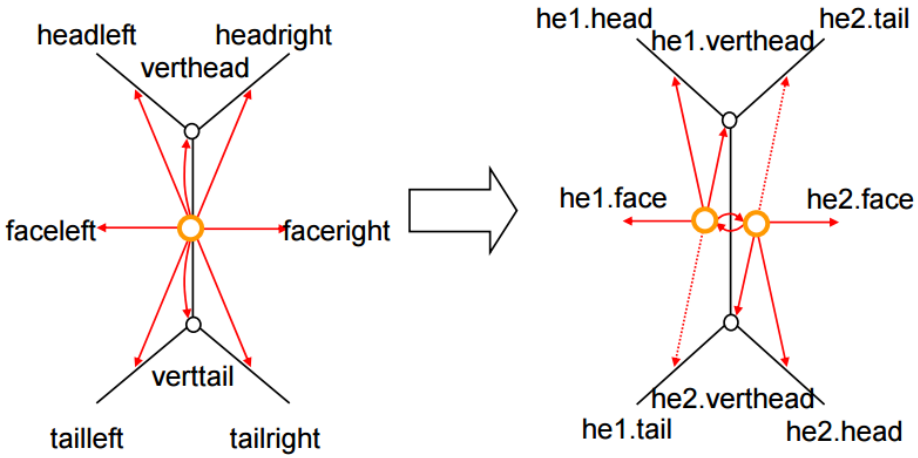
```
class Edge {
   Edge headleft,headright,
        tailleft,tailright;
   Face faceleft,faceright;
   Vertex verhead, vertail;
};

class Face {
   Edge edge;
   // face data
};

class Vertex {
   Edge edge;
   // vertex data
};
```

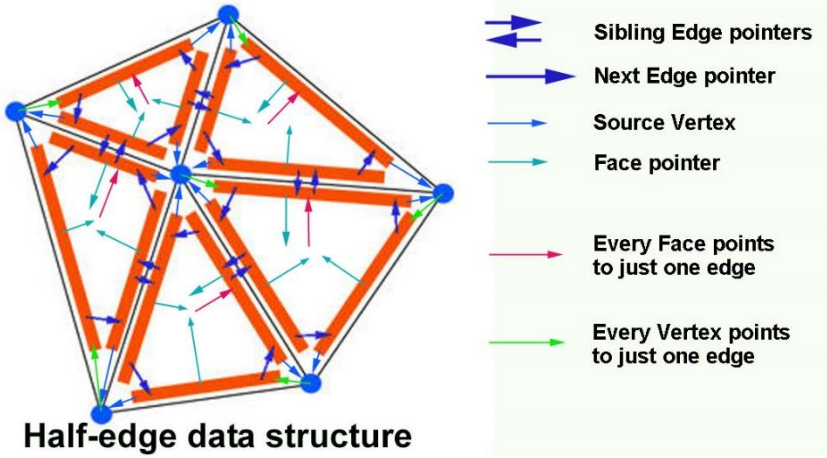
```
// sprehod preko robov poligona
FaceEdges(Face f) {
 e0 = f.edge;
 e = e0;
 do {
   if(e.faceleft== f)
      e = edge.headleft;
   else
      e = edge.tailright;
 } while (e != e0);
```



Half-edge predstavitev

- Podobno kot *winged-edge*, le da rob razdelimo na dva
  - en rob kaže naprej, en nazaj
  - dodamo še kazalec na drug pol-rob, lahko izpustimo kazalec nazaj
- Pohitritev
  - ni *if* stavka pri iskanju robov poligona ali robov oglišča
  - prostor enako cca. 27 dodatnih kazalcev (če ni kazalca nazaj)
- Uporablja Maya in podobna orodja

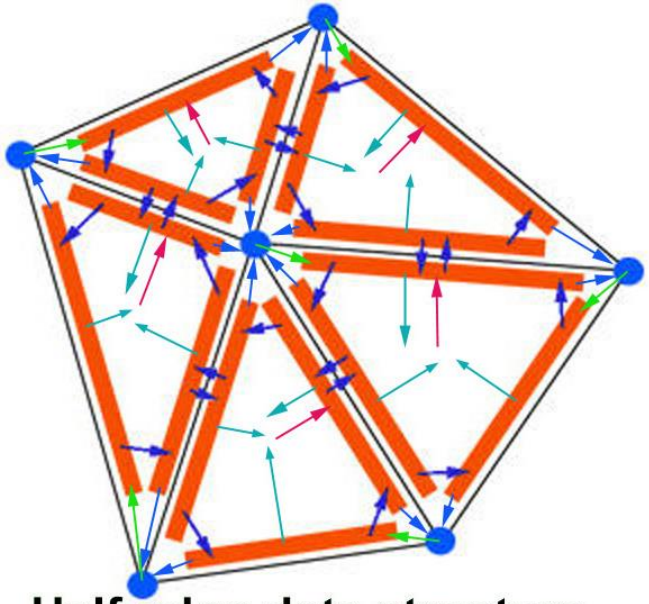




Half-edge primer

```
struct HalfEdge {
   HalfEdge* head, *tail;
   HalfEdge* opposite;
   Face* face;
   Vertex* verthead;
};


struct Face {
   Edge* edge;
   // face data
};


struct Vertex {
   Edge* edge;
   // vertex data
};
```

```
FaceEdges(Face *f) {
 e0 = f->halfedge;
 e0 = e;
   do {
     e = e->head;
   }
   while (e != e0);
}
```



Half-edge data structure

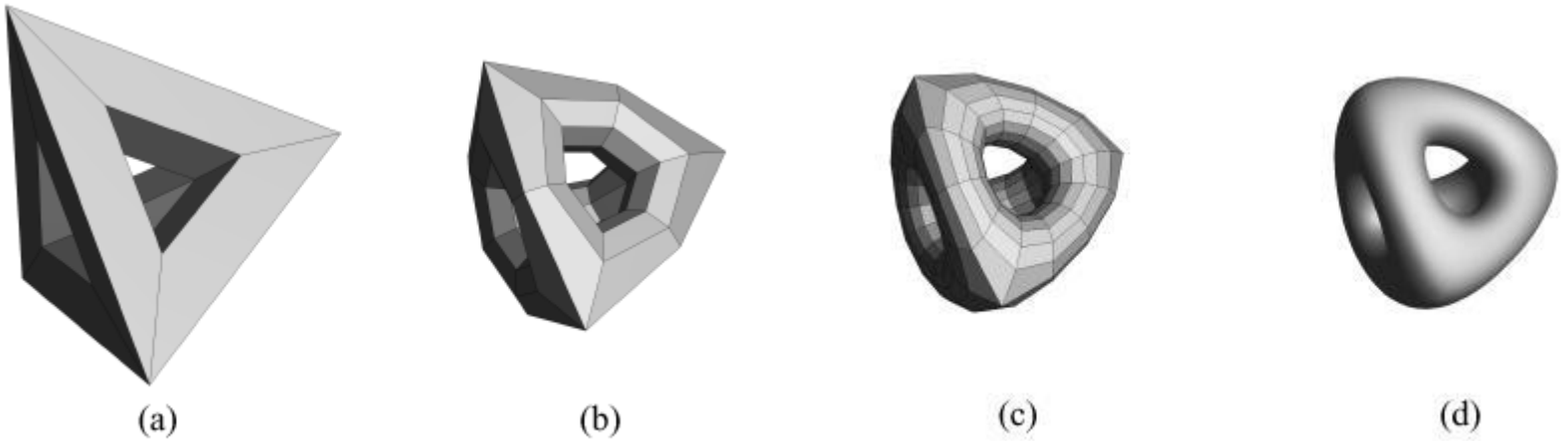
Deljene ploskve

Deljene ploskve (*subdivision surfaces*)

- Problem poligonskih mrež je grobost oz. **nezveznost**
- **Osnovna ideja**: kako lahko iz grobega približka naredimo gladko/zvezno krivuljo?
  - z delitvijo posameznih odsekov

Deljene ploskve

- Grobo poligonsko mrežo iterativno razdeljujemo
  - limitira k zvezni ploskvi (deljenje ustavimo, ko dobimo želeno zveznost)



Deljene ploskve

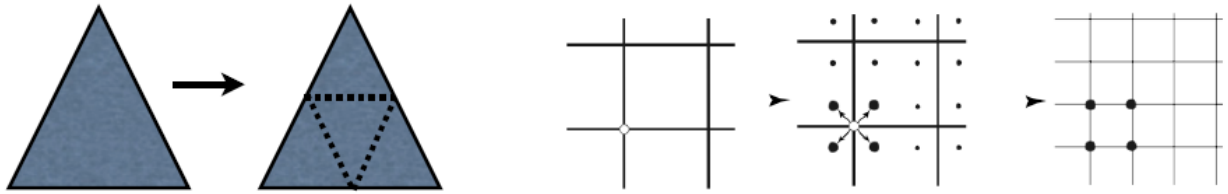
- Se veliko uporabljajo v animaciji/filmih – npr. večina Pixarjevih karakterjev je narejena z deljenimi ploskvami
  - enostaven opis kompleksnih ploskev
  - poljubna topologija
  - podpirajo lokalne spremembe
  - zagotovljena zveznost
  - podpirajo več nivojev ločljivosti – lahko nadziramo deljenje

Poligonske mreže vs. deljene ploskve

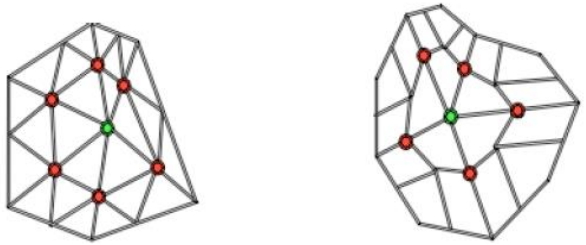
	Poligonske mreže	Deljene ploskve
natančne	Ne	Da
zgoščene	Ne	Da
lokalne	Da	Da
poljubna topologija	Da	Da
zvezne	Ne	Da
učinkovit prikaz	Da	Da
učinkoviti preseki	Ne	Ne

Kako delimo?

- Algoritmov je več, prvi nastali že v 70. letih, delijo se po različnih kriterijiih:
  - delitev poligonov vs. delitev oglišč



  - tip mreže: trikotniki vs. štirikotniki

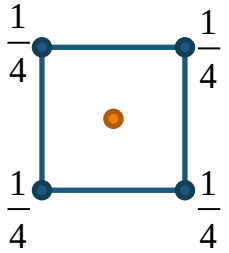


  - zveznost limite – zvezen prvi odvod - C1 ali zvezen drugi odvod - C2
  - aproksimacija (premakne orig. oglišča) vs. interpolacija (jih ne premakne)

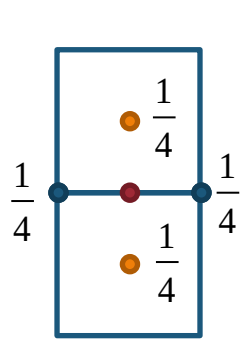
Deljenje Catmull-Clark (1978)

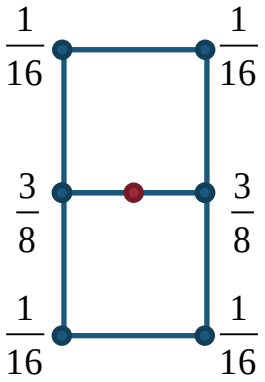
- Deli štirikotnike z aproksimacijo in zagotavlja C2 zveznost v notranjosti
- Algoritem dodaja/spreminja oglišča kot uteženo vsoto obstoječih oglišč
- Algoritem:
  - 1. Dodaj nova oglišča v_f v središče vseh štirikotnikov
  - 2. Dodaj novo oglišče v_e za vsak rob kot središče (starih) oglišč roba in (novih) oglišč sosednjih štirikotnikov

(1) novo oglišče v sredini štirikotnika



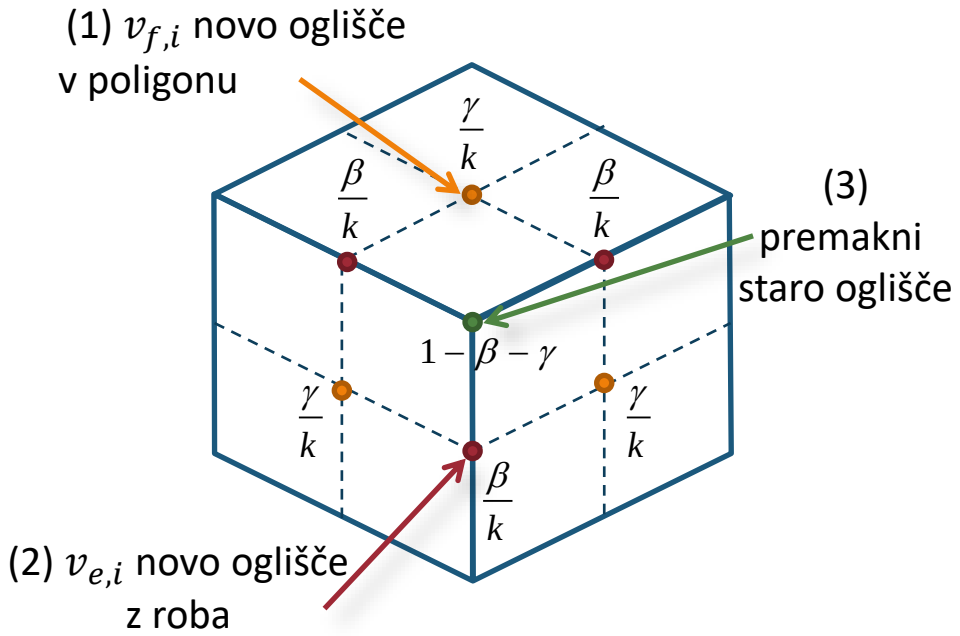
(2) novo oglišče glede na rob (obe varianti sta ekvivalentni)





Deljenje Catmull-Clark

- 3. Premakni originalna oglišča z uteženo vsoto sosednjih novih oglišč
  - $v' = (1 - \beta - \gamma)v + \frac{\beta}{k}\sum_i v_{e,i} + \frac{\gamma}{k}\sum_i v_{f,i}$
    - $\beta = \frac{3}{2k}, \quad \gamma = \frac{1}{4k}$
    - k je število robov v oglišču v
- 4. naredi nove štirikotnike iz izračunanih oglišč
- Vse štiri korake ponovimo do želene globine, vsakič dobimo bolj zvezen predmet



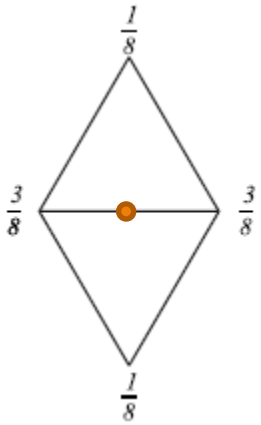
Deljenje Loop (1987)

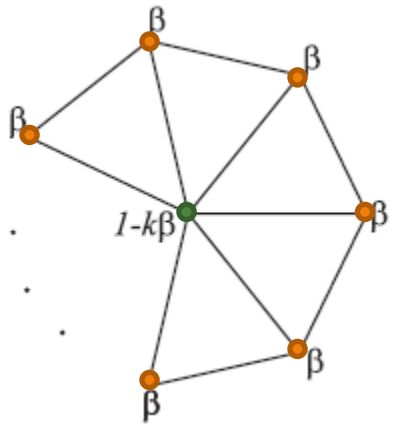
Podobno kot Catmull-Clark, le da za trikotnike

1. dodaj novo oglišče za vsak rob
2. premakni originalna oglišča kot uteženo vsoto novih oglišč

Različne variante obstajajo za izbor uteži β, npr.

$$\beta = \frac{1}{k}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{k}\right)^2\right)$$





REFERENCE

- wiki: *Polygon meshes*
- Pixar: *OpenSubDiv Library*
- N. Guid: *Računalniška grafika*, FERI Maribor
- J.D. Foley, A. Van Dam et al.: *Computer Graphics: Principles and Practice in C*, Addison Wesley
- P. Shirley, S. Marschner: *Fundamentals of Computer Graphics*, A.K. Peters