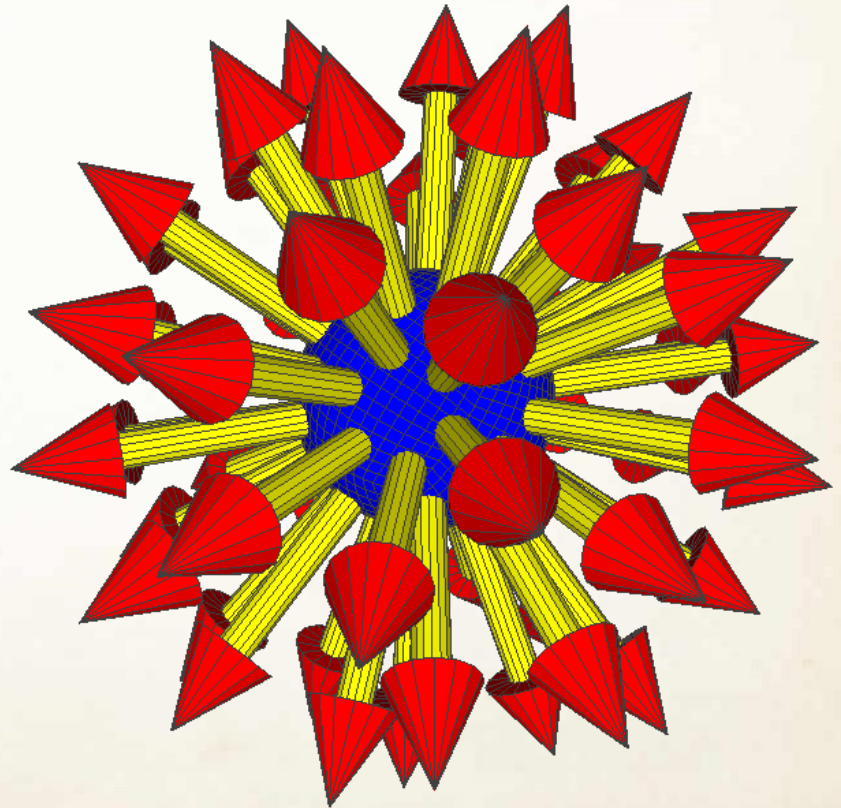


Duomenų struktūros ir algoritmai

12 paskaita

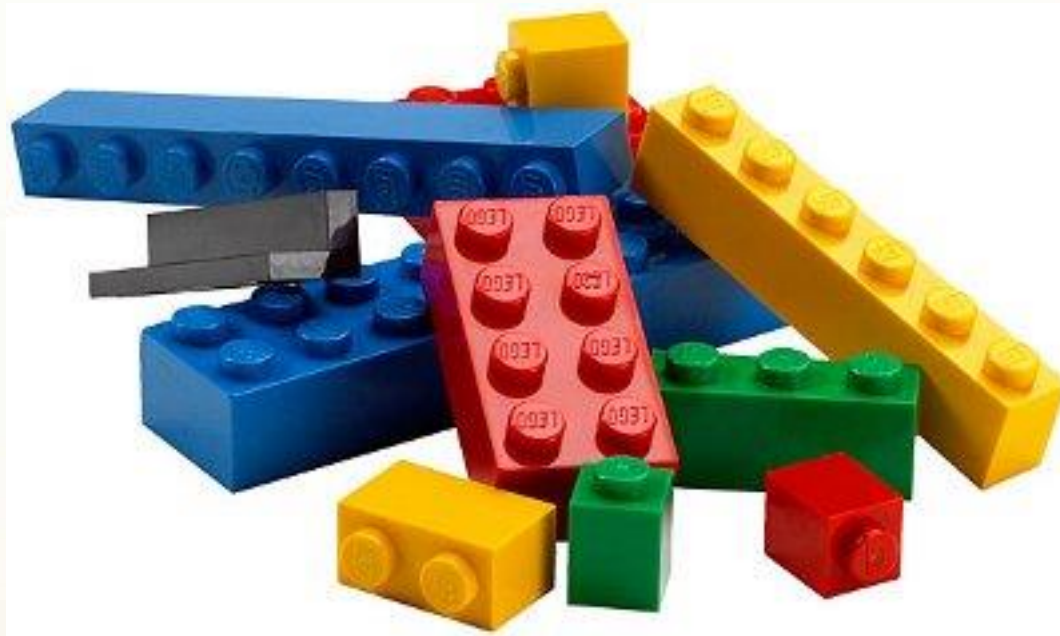
2020-04-29



Svarbi informacija

- Gegužės 6 d. paskaita skirta kursui užbaigti.
- Gegužės 13 d. paskaitos nebus (dėl konferencijos Rumunijoje).
- Gegužės 20 d. paskaita skirta Jūsų pristatymams.
- Gegužės 27 d. paskaita skirta Jūsų 3D modelių pristatymui ir 5 gražiausių išrinkimui (+0,25 balo premija 5 išrinktiesiems modelių autoriams).

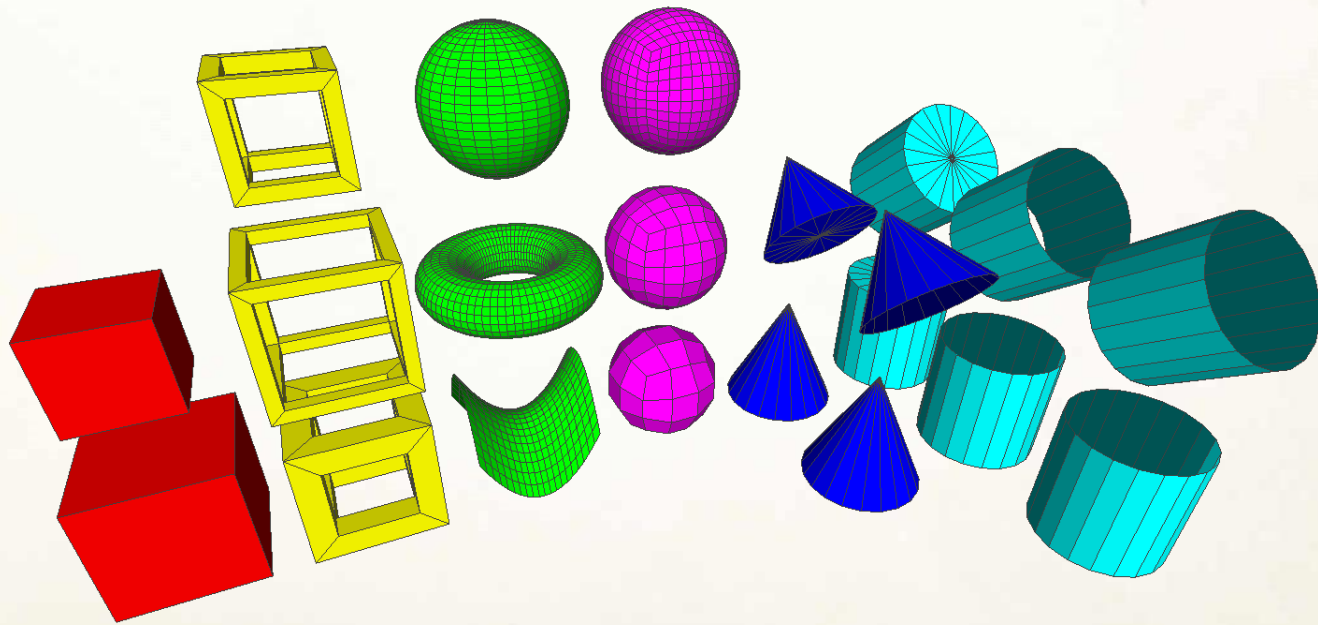
Norint kažką sukonstruoti, reikia...



turėti detalių.

12 paskaitos tikslas

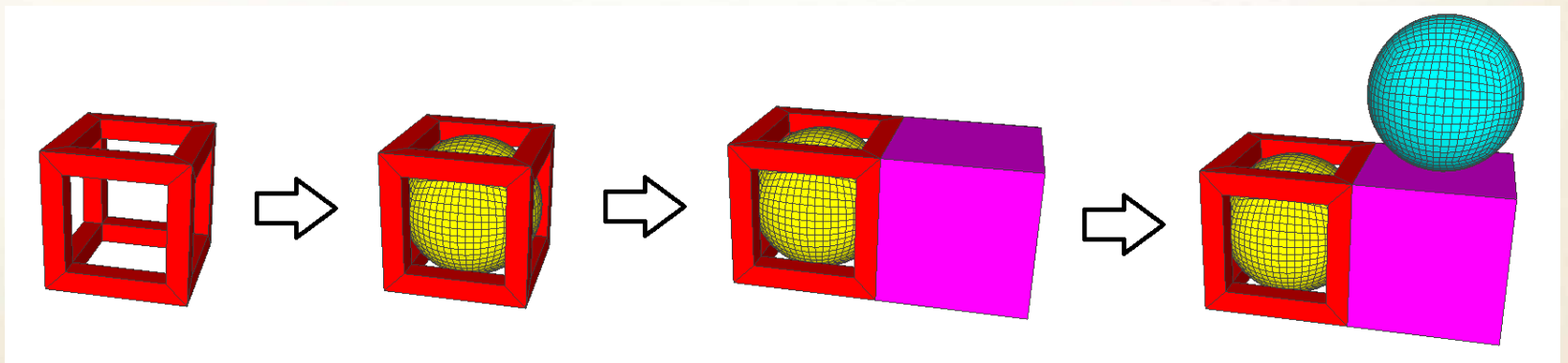
- Susipažinti su *python* modulio ***add.py*** 1.2 versija.
- Sukurti skaitmeninį modelį naudojantis šiuo moduliu:



- Įkelti sukurtą 3D modelį į <https://sketchfab.com> svetainę.

*python modulis **add.py***

- Modulis kiekvienais metais atnaujinamas!
- Modulis pritaikytas kurti 3D skaitmeninius modelius **OFF** formatu.
- Būtina Python 3 versija!
- Modelio kūrimas vyksta konstravimo principu:



Pagrindinė idėja

Į sąrašą *vertices* įrašomos viršūnių koordinatės *string* pavidalu, į sąrašą *faces* įrašoma informacija apie kiekvieną 3D modelio sieną irgi *string* pavidalu. Šios informacijos užtenka norint sugeneruoti 3D modelį OFF formatu.



```
vertices = []
faces = []

<...>

def off(mesh):
    global vertices, faces
    file = open(mesh, 'w')
    file.write('%s\n%d %d %d\n' % ('OFF', len(vertices), len(faces), 0))
    for i in range(len(vertices)):
        file.write('%s\n' % vertices[i])
    for j in range(len(faces)):
        file.write('%s\n' % faces[j])
    file.close()
```

*Modulio **add.py** funkcijos*

- **def cube(c,e,RGB):** # c = center, e = edge width
- **def cube2(c,e,b,RGB):** # c = center, b = border width, e = edge width
- **def parametric(S,min_u,max_u,grid_u,min_v,max_v,grid_v,RGB):**
S - parametric uv surface, grid - detail, RGB - color
- **def sphere(c,r,k,RGB):** # c - center, r - radius, k - detail, RGB - color
- **def cylinder(A,B,r,k,RGB):** # A - start point, B - end point, r - radius, k - detail, RGB - color
- **def cylinder2(A,B,r,k,RGB):** # A - start point, B - end point, r - radius, k - detail, RGB - color
- **def cylinder3(A,B,r,k,RGB):** # A - start point, B - end point, r - radius, k - detail, RGB - color
- **def cone(A,B,r,k,RGB):** # A - start point, B - end point, r - radius, k - detail, RGB - color
- **def cone2(A,B,r,k,RGB):** # A - start point, B - end point, r - radius, k - detail, RGB - color
- **def off(mesh):** # mesh – off file

*Modulio **add.py** 1.1 versija*

- **def newface(A,RGB):** # A = set of 3D points
- **def pyramid(c,e,h,RGB):** # c - center, e - edge width, h - high
- **def rectangle3D(c,e,RGB):** # c - center, e - width of edges
- **def circle(A,B,r,k,RGB):** # A - start point, B - end point, r - radius, k - detail
- **def spin3D(A,B,S,min_t,max_t,grid_t,k,RGB):** # A - start point, B - end point, r - radius, k - detail, RGB - color, S – parametric function

*Modulio **add.py** 1.2 versija*

- **def axes(C):** # adds 3D axes to default layer, C - center
- **def layer():** # returns 3D mesh from default layer and deletes default layer
- **def mesh(M):** # adds 3D mesh to default layer, M – selected layer
- **def center(M):** # returns 3D center from selected layer M
- **def rotateX(M,angle,P):** # rotates selected layer M around X-axis,
angle – rotation angle, P – point of rotation center
- **def rotateY(M,angle,P):** # rotates selected layer M around Y-axis
- **def rotateZ(M,angle,P):** # rotates selected layer M around Z-axis
- **def move(M,V):** # moves selected layer M according to vector V
- **def zoom(M,s):** # zooms selected layer M by scale s
- **def merge(M):** # merges selected layers M=[M[0],M[1], ...]
- **def load(mesh):** # loads mesh from off file
- **def color(M,RGB):** # sets RGB color to layer M

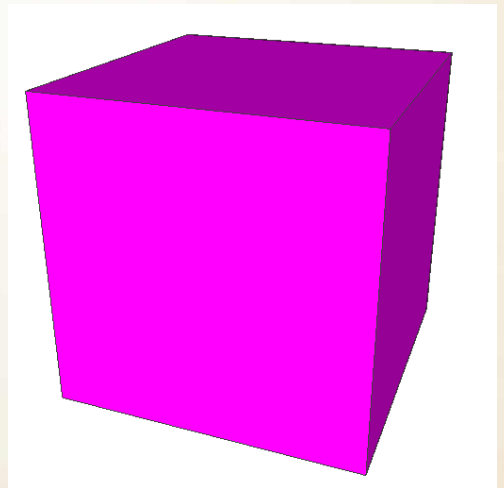
def cube(c, e, RGB)

```
def cube(c,e,RGB): # c - center, e - edge width, RGB - color
    global vertices, faces
    F = [[0,4,5,1],[0,1,3,2],[0,2,6,4],[1,5,7,3],[2,3,7,6],[4,6,7,5]]
    V = [[0,0,0],[0,0,e],[0,e,0],[0,e,e],[e,0,0],[e,0,e],[e,e,0],[e,e,e]]
    for i in range (0,6):
        faces += ['4 '+str(F[i][0]+len(vertices))+ ' '+str(F[i][1]+len(vertices))+
            ' '+str(F[i][2]+len(vertices))+ ' '+str(F[i][3]+len(vertices))+
            ' '+str(RGB[0])+ ' '+str(RGB[1])+ ' '+str(RGB[2])]
    for j in range (0,8):
        vertices += [str(c[0]+V[j][0]-e/2)+' '+str(c[1]+V[j][1]-e/2)+
            ' '+str(c[2]+V[j][2]-e/2)]
```

c – centro 3D koordinatės, e – briaunos ilgis,
RGB – kubo spalva.

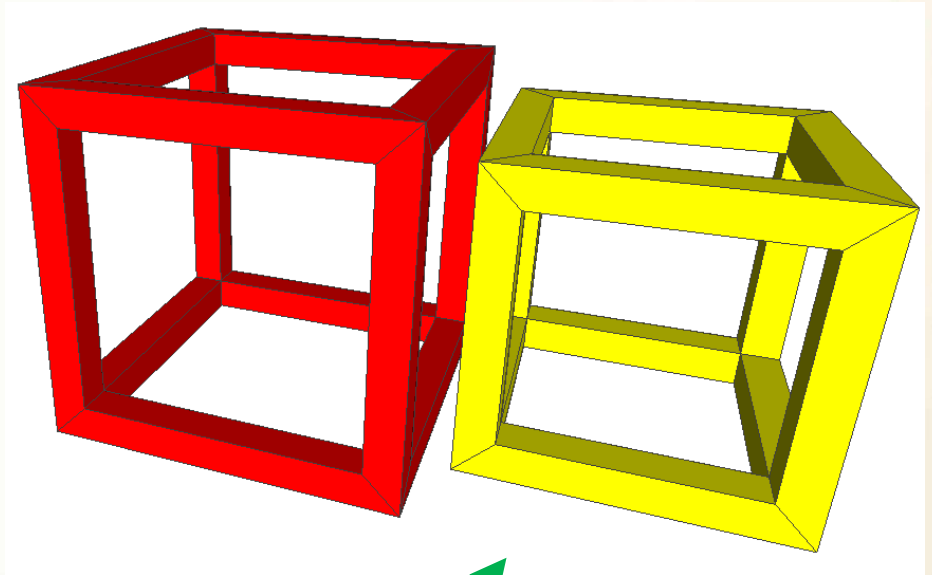
cube.py generuoja cube.off failą:

```
1 import add
2 add.cube([1,0,0],1,[255,0,255])
3 add.off('cube.off')
```



def cube2(c, e, b, RGB)

c – centro 3D koordinatės,
e – briaunos ilgis,
b – briaunos storis,
RGB – kubo spalva.

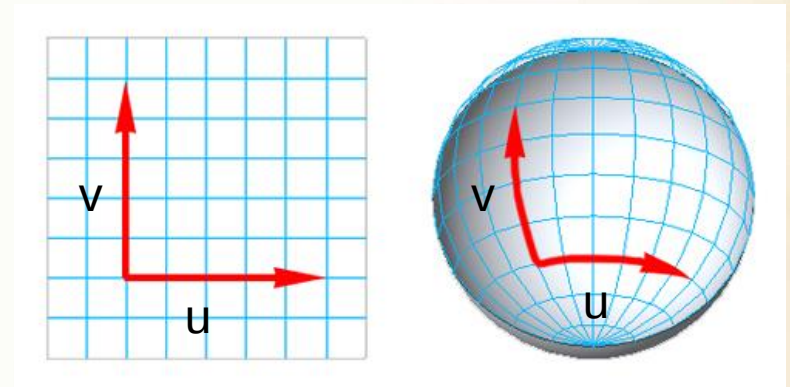


cube2.py generuoja cube2.off failą:

```
1 import add
2 add.cube2([1,0,0],1,0.1,[255,0,0])
3 add.cube2([2,0,0],0.8,0.1,[255,255,0])
4 add.off('cube2.off')
```

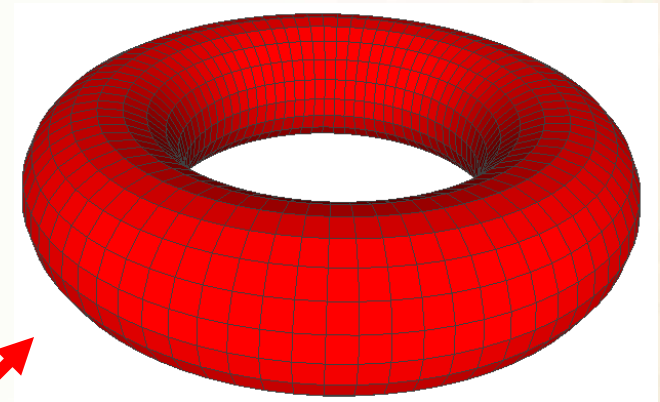
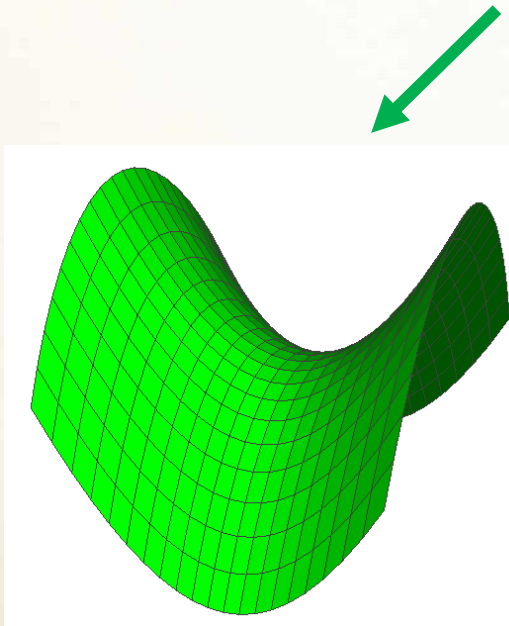
def parametric(S, min_u, max_u, grid_u, min_v, max_v, grid_v, RGB)

S – parametrinio uv paviršiaus f-ja,
min_u – parametro u mažiausia reikšmė,
max_u – parametro u didžiausia reikšmė,
grid_u – paviršiaus detalumas u atžvilgiu,
min_v – parametro v mažiausia reikšmė,
max_v – parametro v didžiausia reikšmė,
grid_v – paviršiaus detalumas v atžvilgiu,
RGB – paviršiaus spalva.



Parametrinių paviršių pavyzdžiai

```
1 import add
2 def saddle(u,v):
3     x = u
4     y = v*v - u*u
5     z = -v
6     return ([x, y, z])
7 add.parametric(saddle,-1,1,20,-1,1,20,[0,255,0])
8 add.off('saddle.off')
```

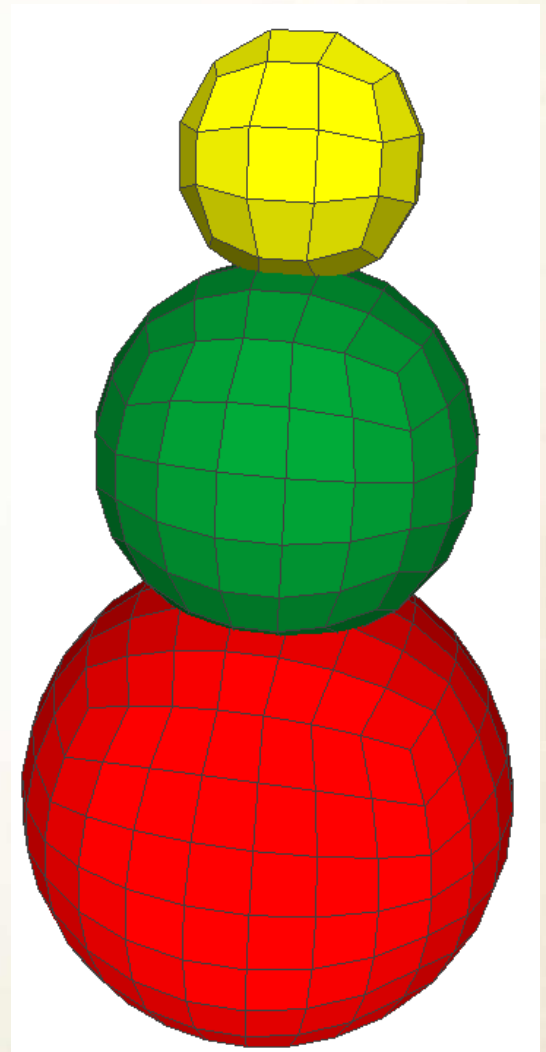


```
1 import add
2 import math
3 a = 3
4 b = 1
5 def torus(u,v):
6     x = (a+b*math.cos(u))*math.cos(v)
7     y = b*math.sin(u)
8     z = (a+b*math.cos(u))*math.sin(v)
9     return ([x, y, z])
10 add.parametric(torus,0,2*math.pi,20,0,2*math.pi,80,[255,0,0])
11 add.off('torus.off')
```

def sphere(c, r, k, RGB)

c – centro 3D koordinatės,
r – spindulio ilgis,
k – sferos detalumas,
RGB – sferos spalva.

```
1 import add
2 add.sphere([0,0.72,0],0.3,3,[255,255,0])
3 add.sphere([0,0,0],0.5,5,[0,153,51])
4 add.sphere([0,-1,0],0.7,7,[255,0,0])
5 add.off('LTsenis.off')
```

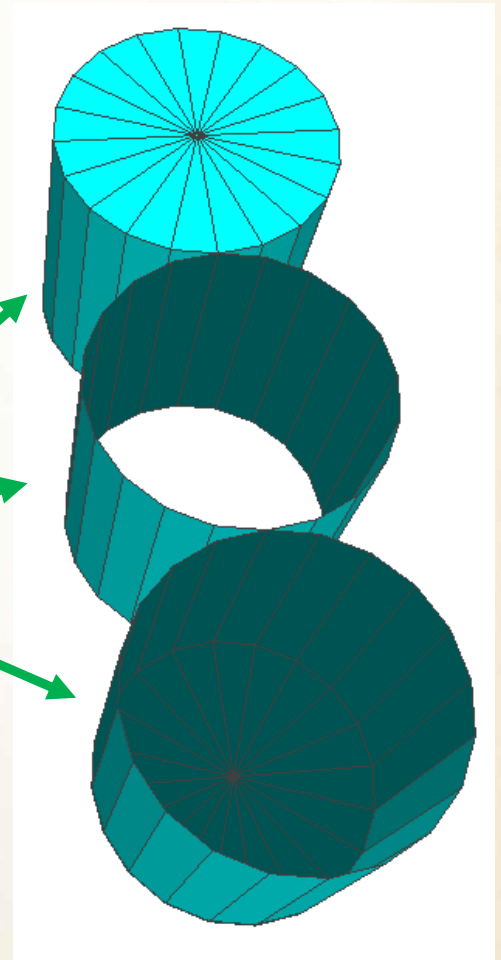


def cylinder(A, B, r, k, RGB) – uždaras cilindras
def cylinder2(A, B, r, k, RGB) – atviras cilindras
def cylinder3(A, B, r, k, RGB) – pusiau atviras cilindras

A – cilindro centro pradžios taškas,
B – cilindro centro pabaigos taškas,
r – cilindro spindulio ilgis,
k – cilindro detalumas,
RGB – cilindro spalva.

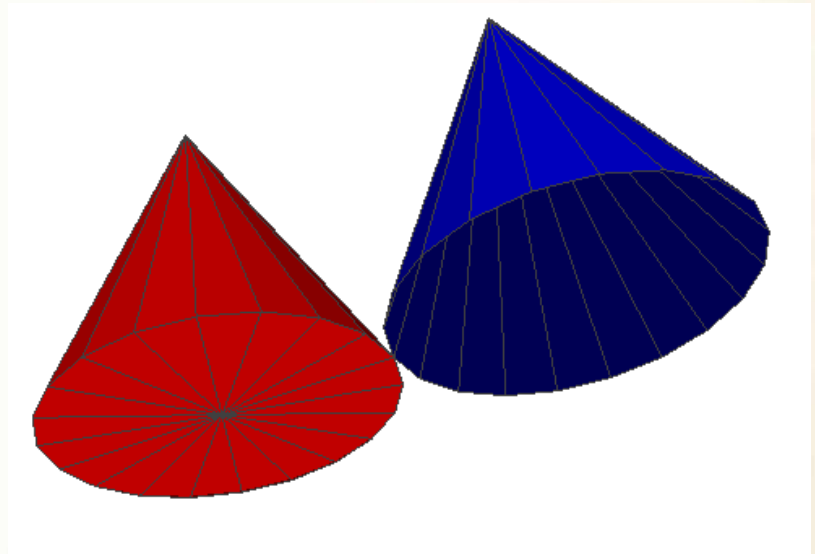
uždaras cilindras
atviras cilindras
pusiau atviras cilindras

```
1 import add  
2 add.cylinder([0,-0.5,0],[0,0.5,0],0.5,20,[0,255,255])  
3 add.cylinder2([0,-0.5,1],[0,0.5,1],0.5,20,[0,255,255])  
4 add.cylinder3([0,-0.5,2],[0,0.5,2],0.5,20,[0,255,255])  
5 add.off('cylinders.off')
```



def cone(A, B, r, k, RGB) – uždaras kūgis
def cone2(A, B, r, k, RGB) – kūgio šoninis paviršius

A – kūgio pagrindo centras,
B – kūgio viršūnė,
r – kūgio pagrindo spindulio ilgis,
k – kūgio detalumas,
RGB – kūgio spalva.

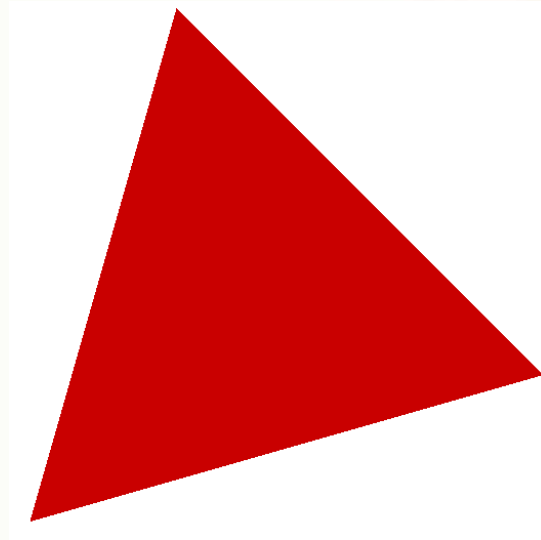


```
1 import add
2 add.cone([0, -0.5, 0], [0, 0.5, 0], 0.5, 20, [255, 0, 0])
3 add.cone2([0, -0.5, 1], [0, 0.5, 1], 0.5, 20, [0, 0, 255])
4 add.off('cones.off')
```



def newface(A,RGB)

A – 3D taškų seka,
RGB – sienos spalva.

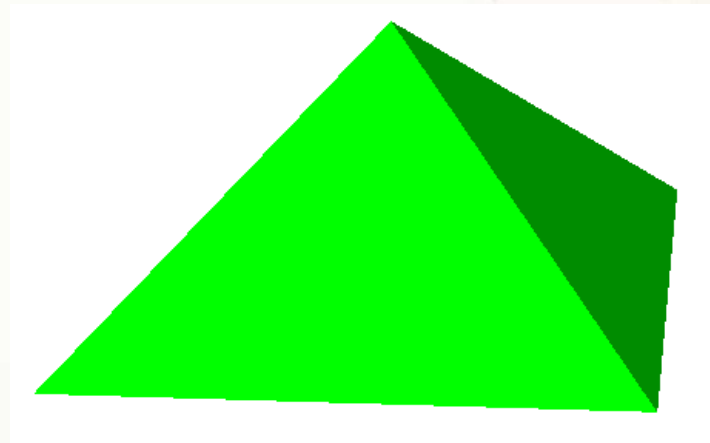


```
1 import add
2 add.newface([[1,0,0],[0,1,0],[0,0,1]],[255,0,0])
3 add.off('trikampis.off')
```

Pastaba: gali būti nebūtinai trikampis.

def pyramid(c,e,h,RGB)

c – kvadrato centro 3D koordinatės,
e – pagrindo (kvadrato) briaunos ilgis,
h – piramidės aukštis,
RGB – piramidės spalva.



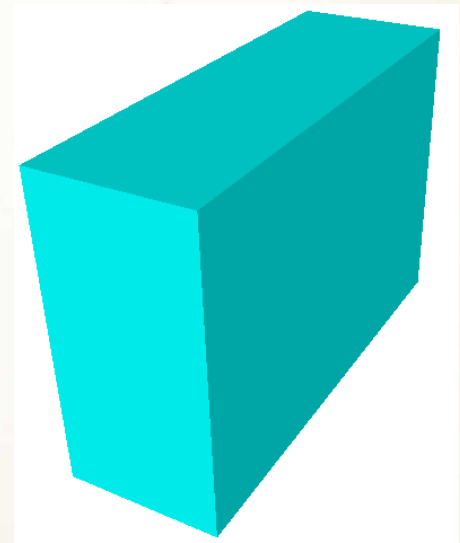
```
1 import add  
2 add.pyramid([0,0,0],1,0.5,[0,255,0])  
3 add.off('pyramid.off')
```

def rectangle3D(c,e,RGB)

c – centro 3D koordinatės,

e – stačiakampio gretasienio briaunų ilgių seka
(atitinkamai X, Y ir Z ašių atžvilgiu),

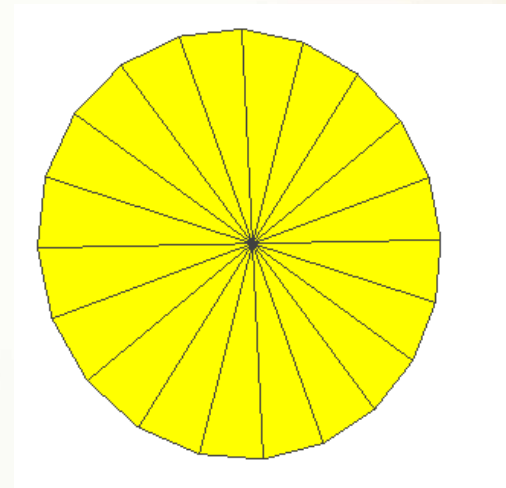
RGB – stačiakampio gretasienio spalva.



```
1 import add
2 add.rectangle3D([0,0,0],[1,2,3],[0,255,255])
3 add.off('plyta.off')
```

def circle(A,B,r,k,RGB)

A – vektoriaus AB pradžios taškas,
B – vektoriaus AB pabaigos taškas,
r – apskritimo spindulys,
k – detalumo parametras,
RGB – apskritimo spalva.

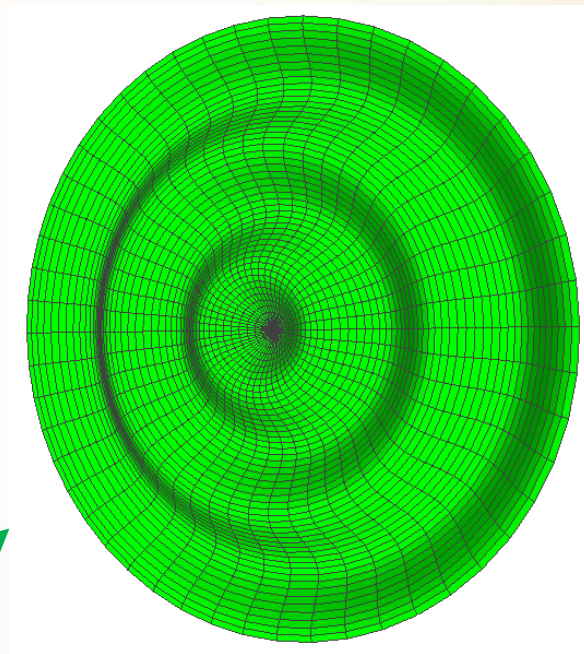


```
1 import add
2 add.circle([0,0,0],[1,1,1],1,20,[255,255,0])
3 add.off('apskritimas.off')
```

Pastaba: vektorius AB statmenas apskritimui, kur A – apskritimo centras.

def spin3D(A,B,S,min_t,max_t,grid_t,k,RGB)

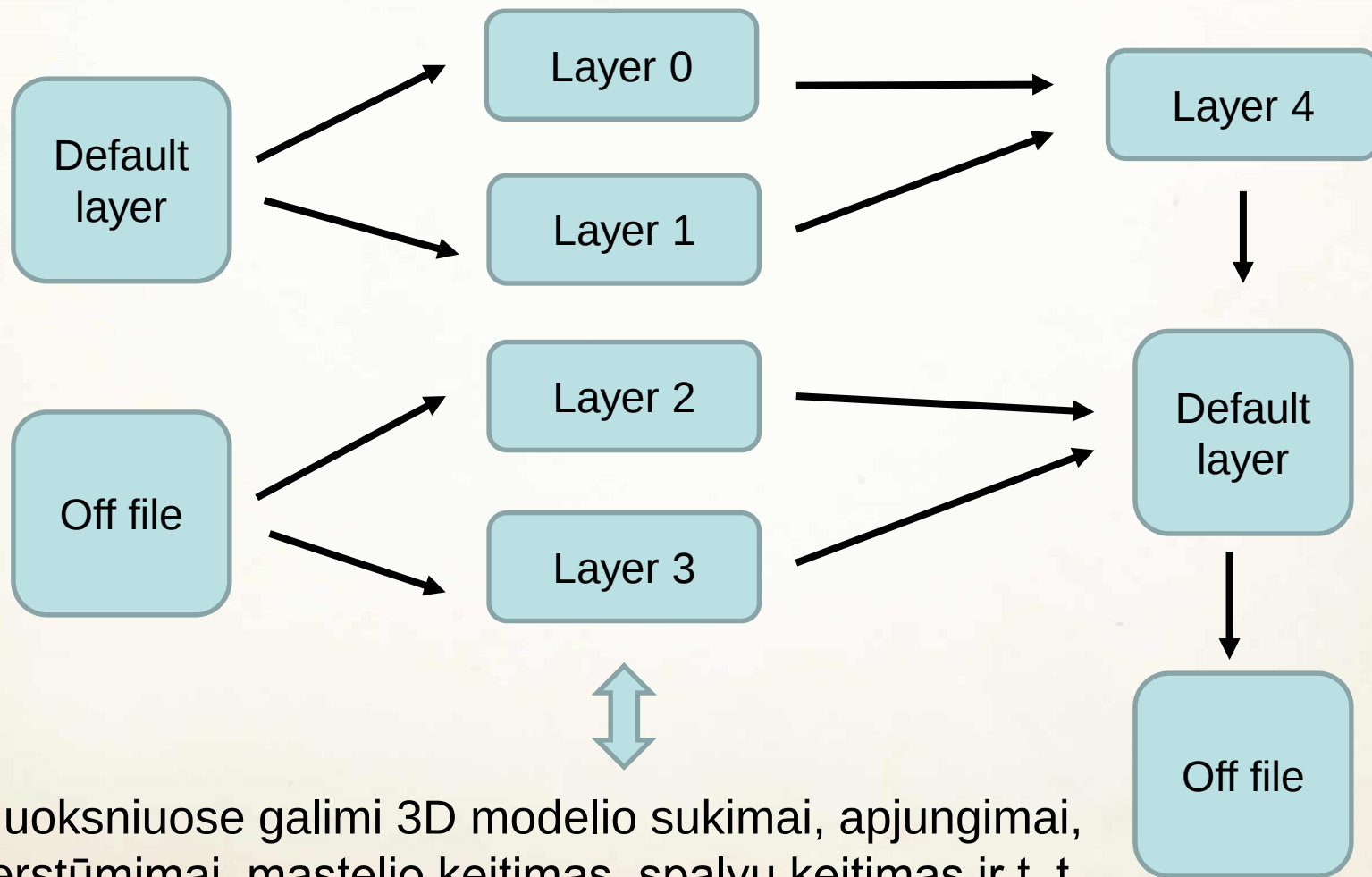
A – vektoriaus AB pradžios taškas,
B – vektoriaus AB pabaigos taškas,
S – parametrinė kreivė,
min_t, max_t – parametrinės kreivės t parametro intervalas,
grid_t – t parametro detalumas,
k – sukinio detalumas,
RGB – sukinio spalva.



```
1 import add
2 import math
3 def curve(t):
4     x = t
5     y = math.cos(t)
6     return ([x, y])
7 add.spin3D([0,0,0],[-1,1,3],curve,0,10*math.pi/2,50,50,[0,255,0])
8 add.off('sukinys.off')
```

Pastaba: kreivė sukama apie vektorių AB, kur A – naujos koordinatų pradžios taškas.

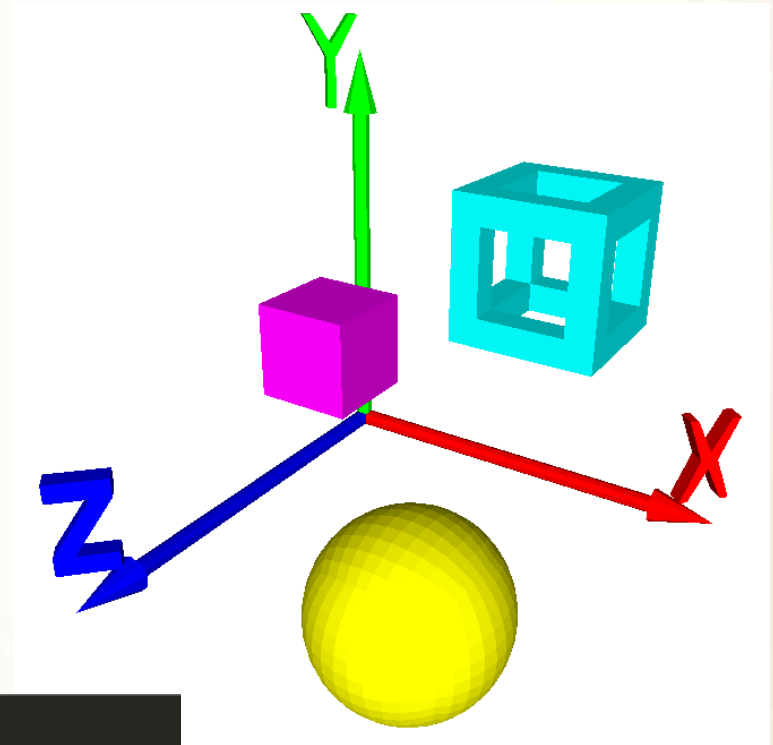
add.py 1.2 versija



Sluoksniuose galimi 3D modelio sukimai, apjungimai, perstūmimai, mastelio keitimas, spalvų keitimas ir t. t.

def axes(C):

C – koordinačių centras,
axes(C) prideda pagalbines 3D
koordinates.



```
import add
add.axes([0,0,0])
add.cube([1,2,2],1,[255,0,255])
add.cube2([3,3,1],1.5,0.3,[0,255,255])
add.sphere([2,-1,2],1.2,10,[255,255,0])
add.off("output4.off")
```

1.2 modulio off.py funkcijos, gražinančios rezultatą sluoksniuose (1)

def layer(): # gražinamas pagrindinio sluoksnio 3D modelis ir ištrinamas iš pagrindinio sluoksnio.

def center(M): # gražinamos pasirinkto 3D modelio M centro koordinatės.

def move(M,V): # gražinamas pastumtas 3D modelis M vektoriumi V.

def zoom(M,s): # gražinamas pakeisto dydžio 3D modelis M masteliu s.

def merge([M[0],M[1], ...]): # sujungiami keli 3D modeliai į vieną.

def load(mesh): # gražinamas 3D modelis iš OFF failo.

def color(M,RGB): # 3D modelis M perdažomas spalva RGB=[R,G,B].

Išimtis:

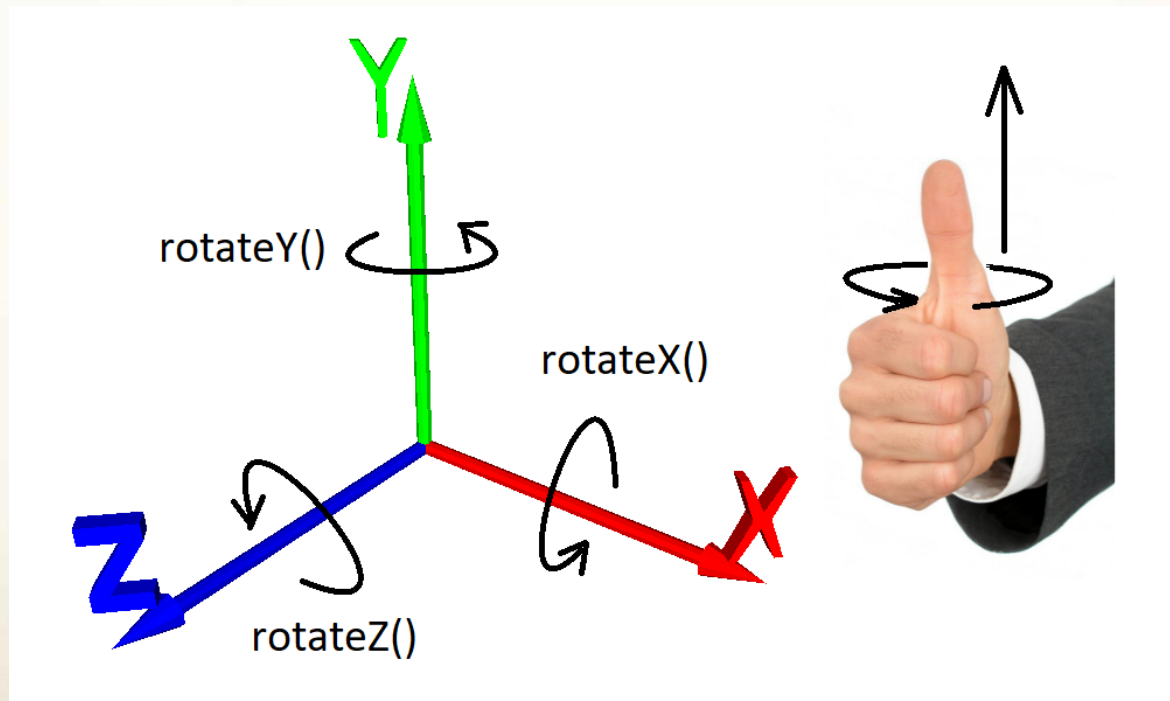
def mesh(M): # 3D modelis M pridedamas prie pagrindinio sluoksnio.

1.2 modulio off.py funkcijos, gražinančios rezultatą sluoksniuose (2)

def rotateX(M,angle,P): # gražinamas pasuktas 3D modelis M apie X
ašį, angle – posūkio kampas, P – posūkio atskaitos taškas

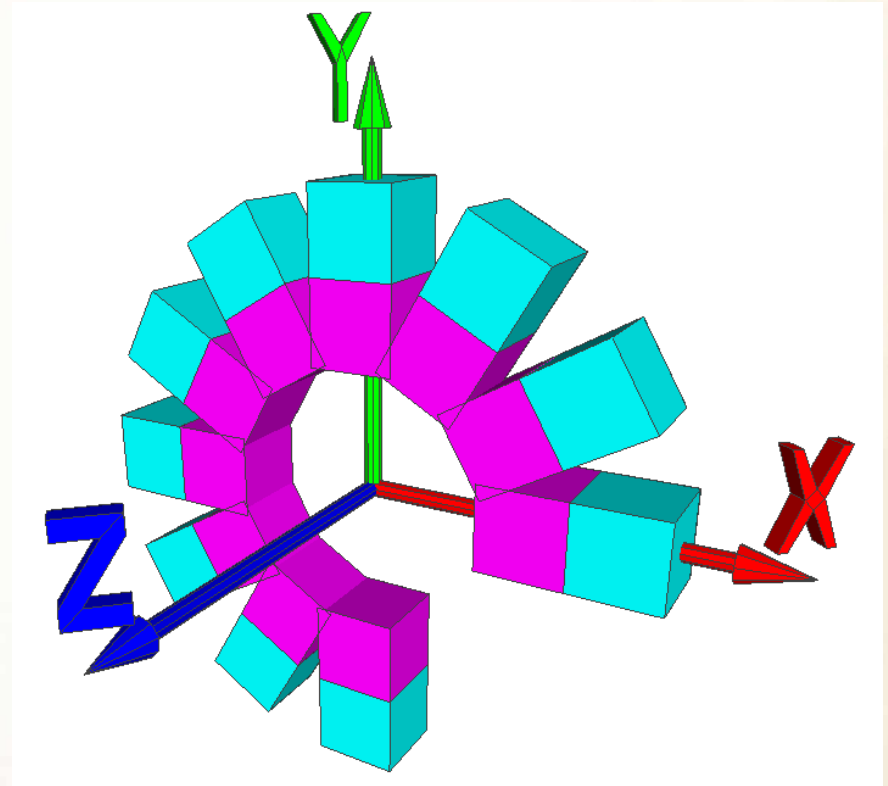
def rotateY(M,angle,P): # analogiškas pasukimas apie Y ašį

def rotateZ(M,angle,P): # analogiškas pasukimas apie Z ašį



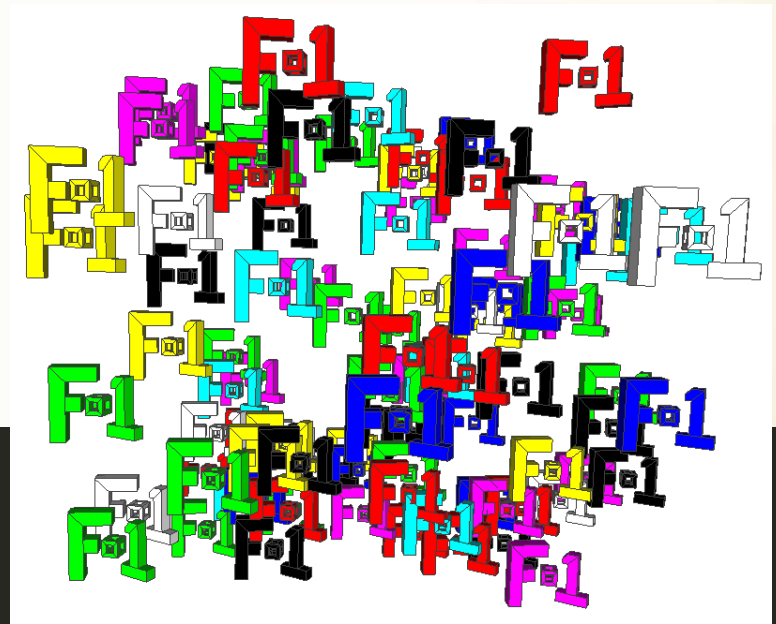
Pavyzdžiai (1)

```
1 import add
2 import math
3 add.cube([2,0,0],1,[255,0,255])
4 add.cube([3,0,0],1,[0,255,255])
5 L = add.layer()
6 add.axes([0,0,0])
7 for i in range(10):
8     M = add.rotateZ(L,i*math.pi/6,[0,0,0])
9     add.mesh(M)
10 add.off('pasukimas.off')
```



Pavyzdžiai (2)

```
1 import add
2 import math
3 import random
4 add.cube2([3.2,2.5,0.5],1.2,0.3,[255,0,0])
5 M1 = add.layer()
6 M2 = add.load("letters/F.off")
7 M2 = add.rotateY(M2,math.pi/10,add.center(M2))
8 M3 = add.load("numbers/1.off")
9 M3 = add.rotateY(M3,-math.pi/10,add.center(M3))
10 M3 = add.move(M3,[3.6,0,0])
11 M3 = add.zoom(M3,0.8)
12 M = add.merge([M1,M2,M3])
13 for i in range(100):
14     M = add.color(M,[random.randint(0,1)*255,random.randint(0,1)*255,random.randint(0,1)*255])
15     add.mesh(add.move(M,[40*random.random(),40*random.random(),40*random.random()]))
16 add.off('failas0.off')
```



Modulio *add.py* 3D modelių pavyzdžiai

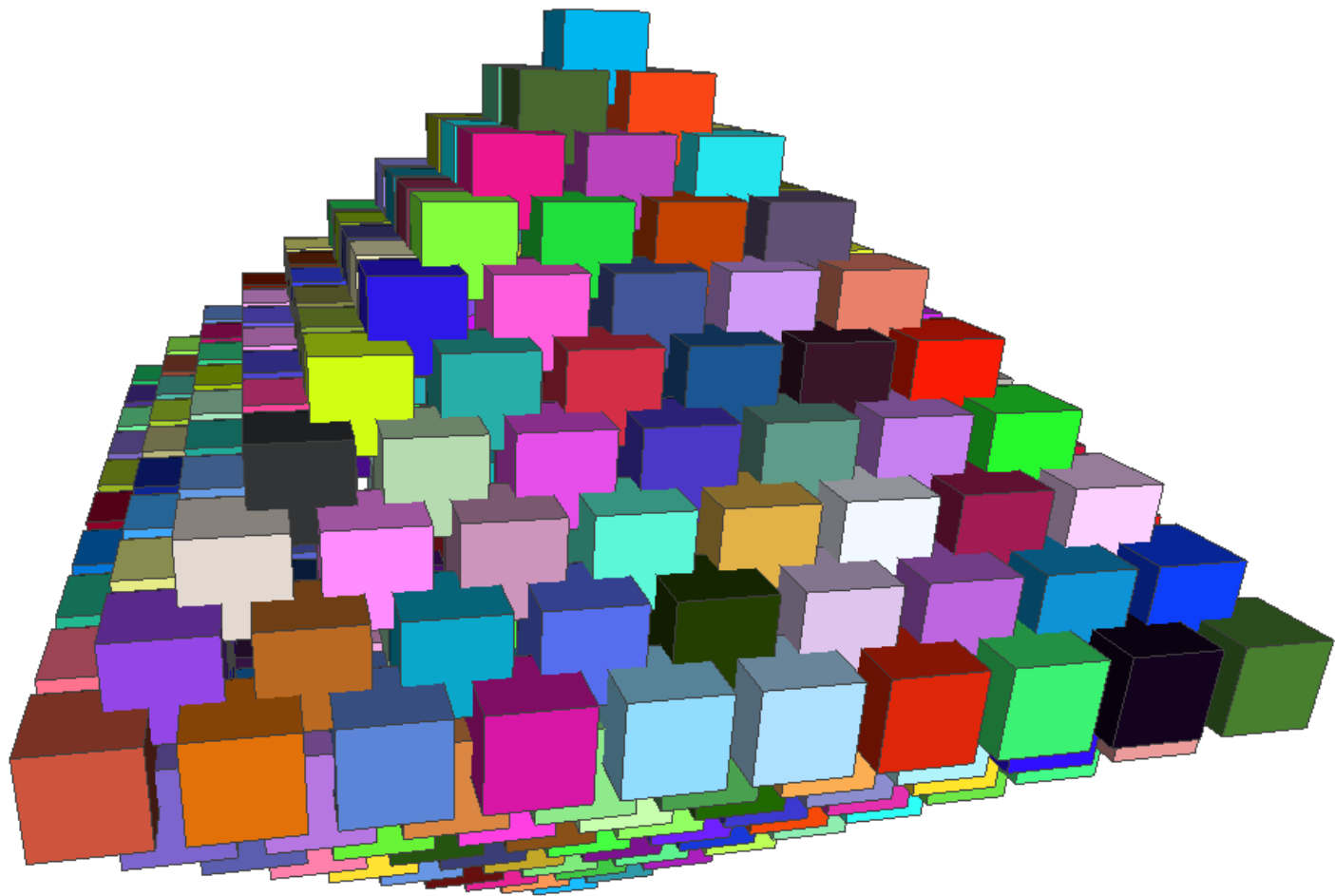
examples.py failas:

```
1 import add
2 add.example1()
3 add.example2()
4 add.example3()
5 add.example4()
6 add.example5()
7 add.example6()
8 add.example7()
9 add.example8()
```

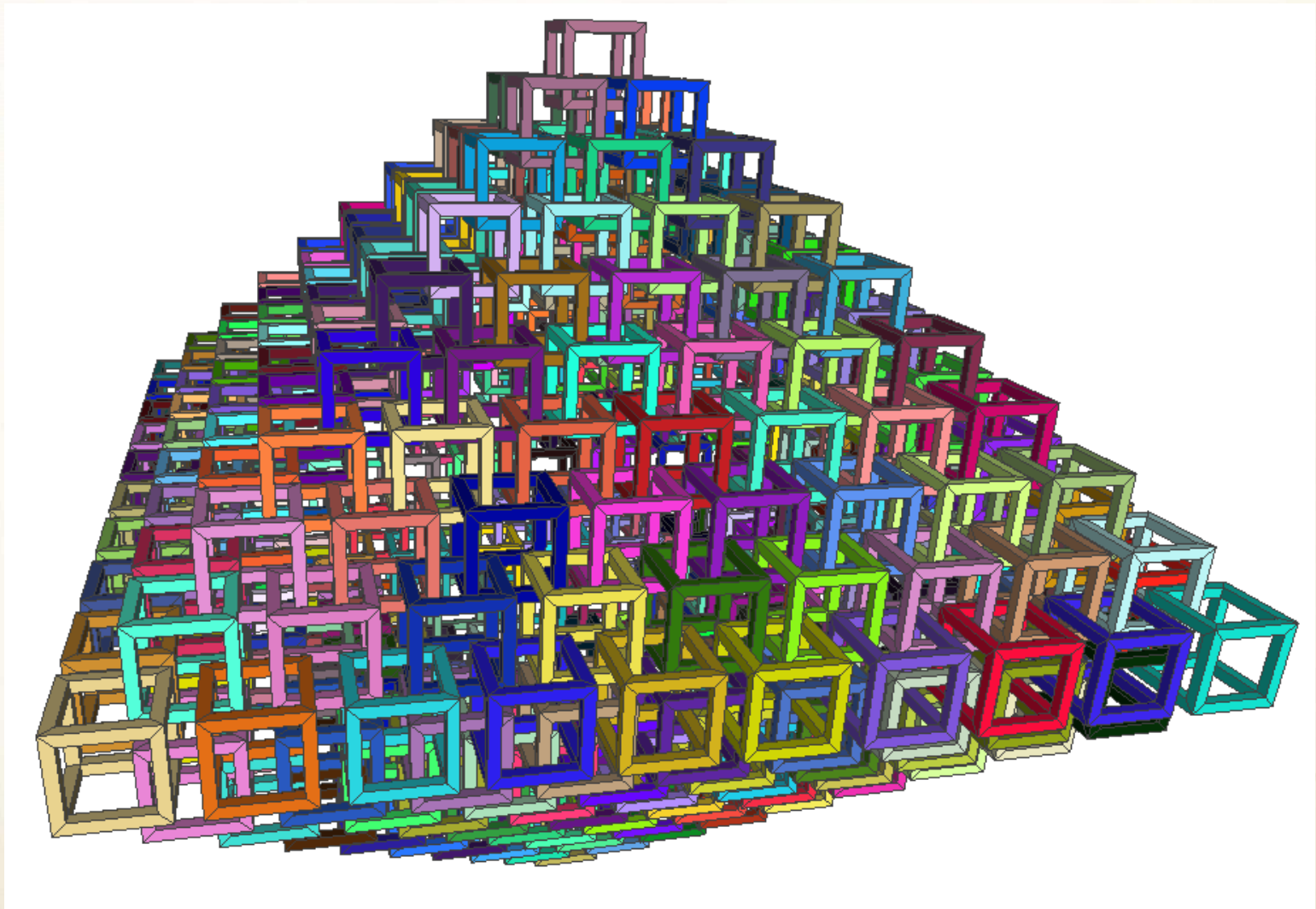


Name	Type	Size
 example1	OFF File	118 KB
 example2	OFF File	970 KB
 example3	OFF File	2 732 KB
 example4	OFF File	2 109 KB
 example5	OFF File	4 723 KB
 example6	OFF File	721 KB
 example7	OFF File	3 465 KB
 example8	OFF File	354 KB

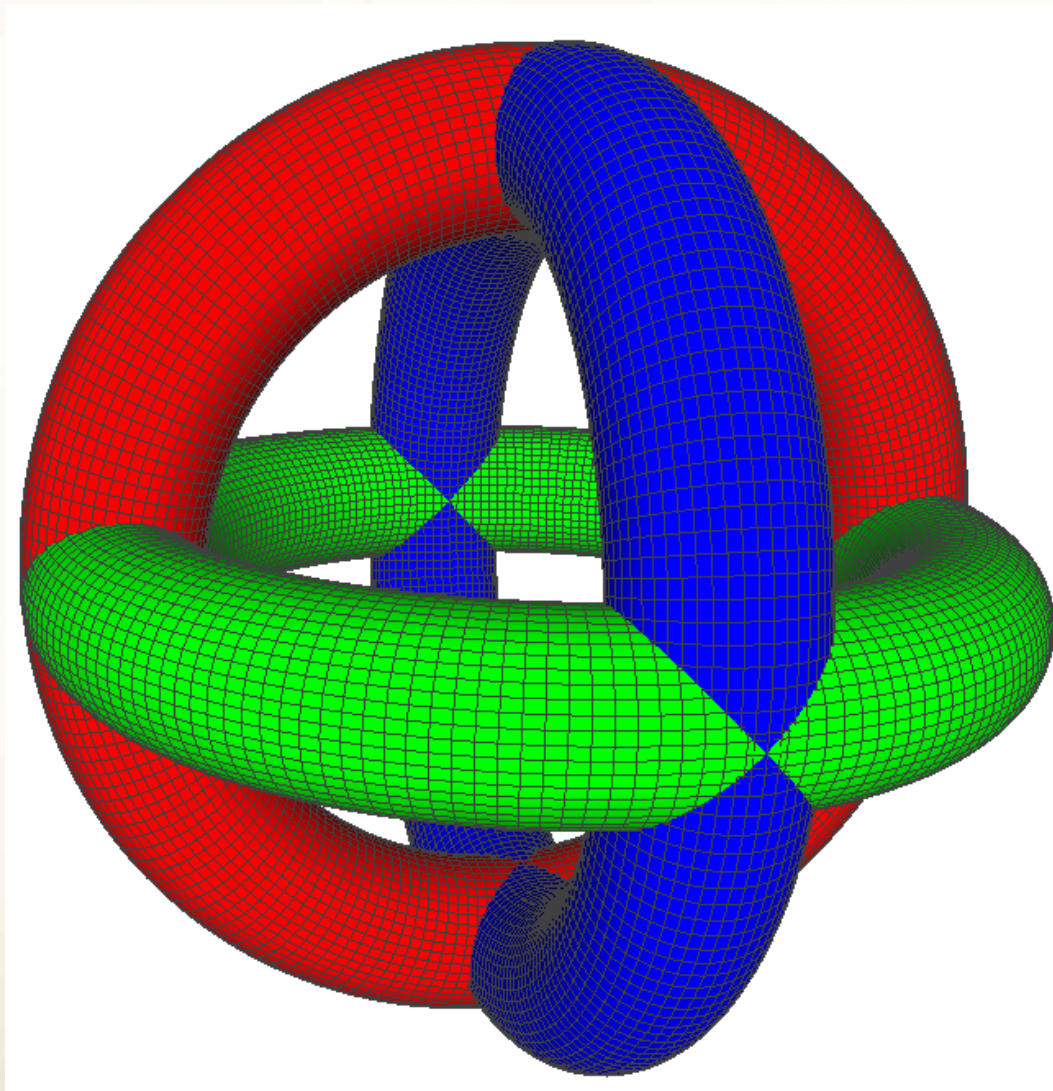
example1.off



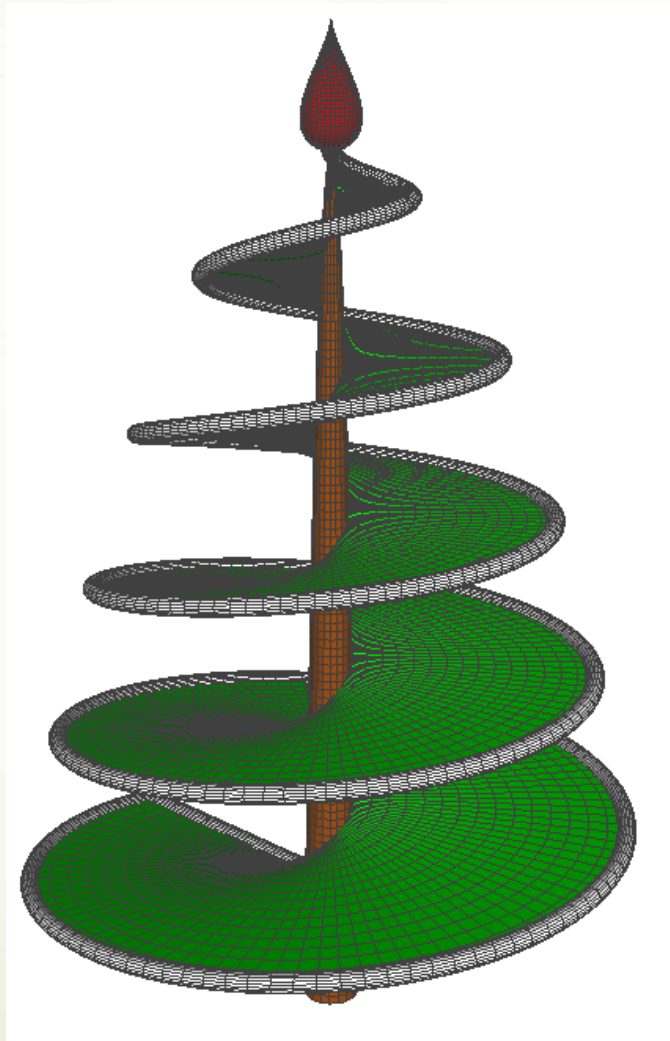
example2.off



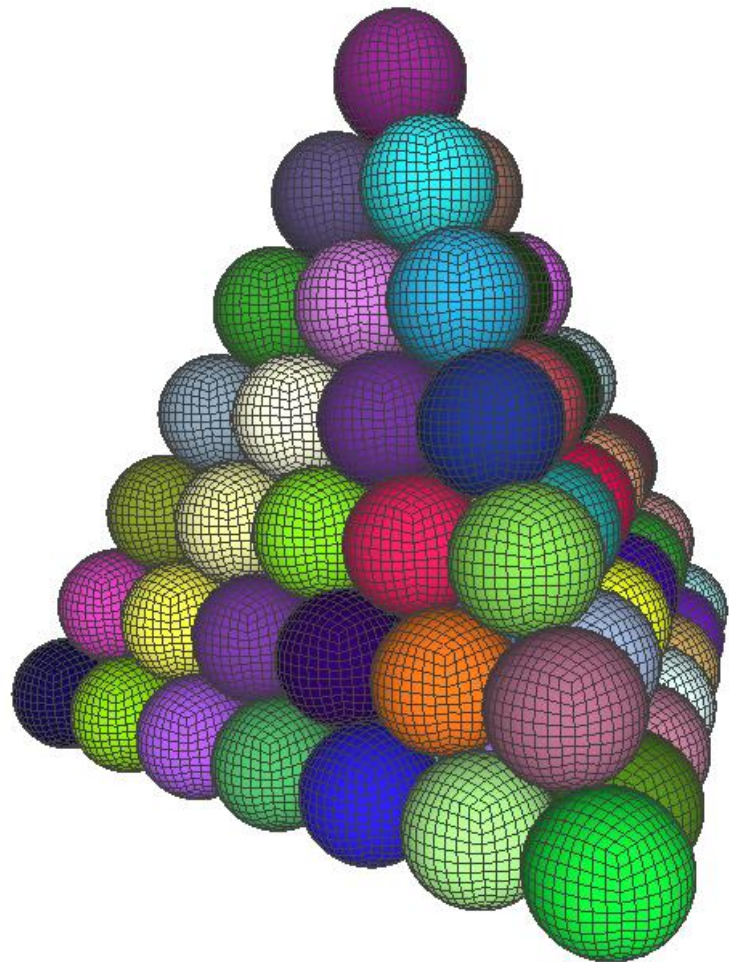
example3.off



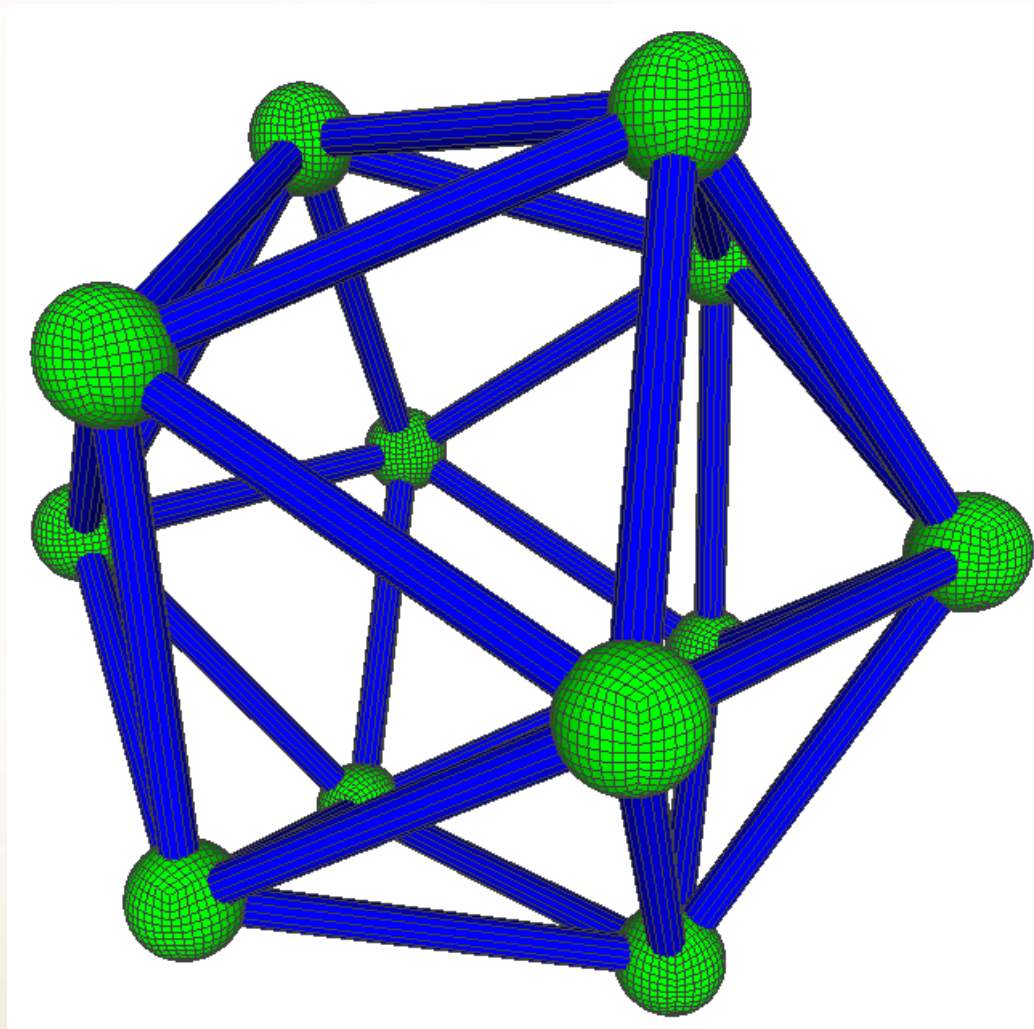
example4.off



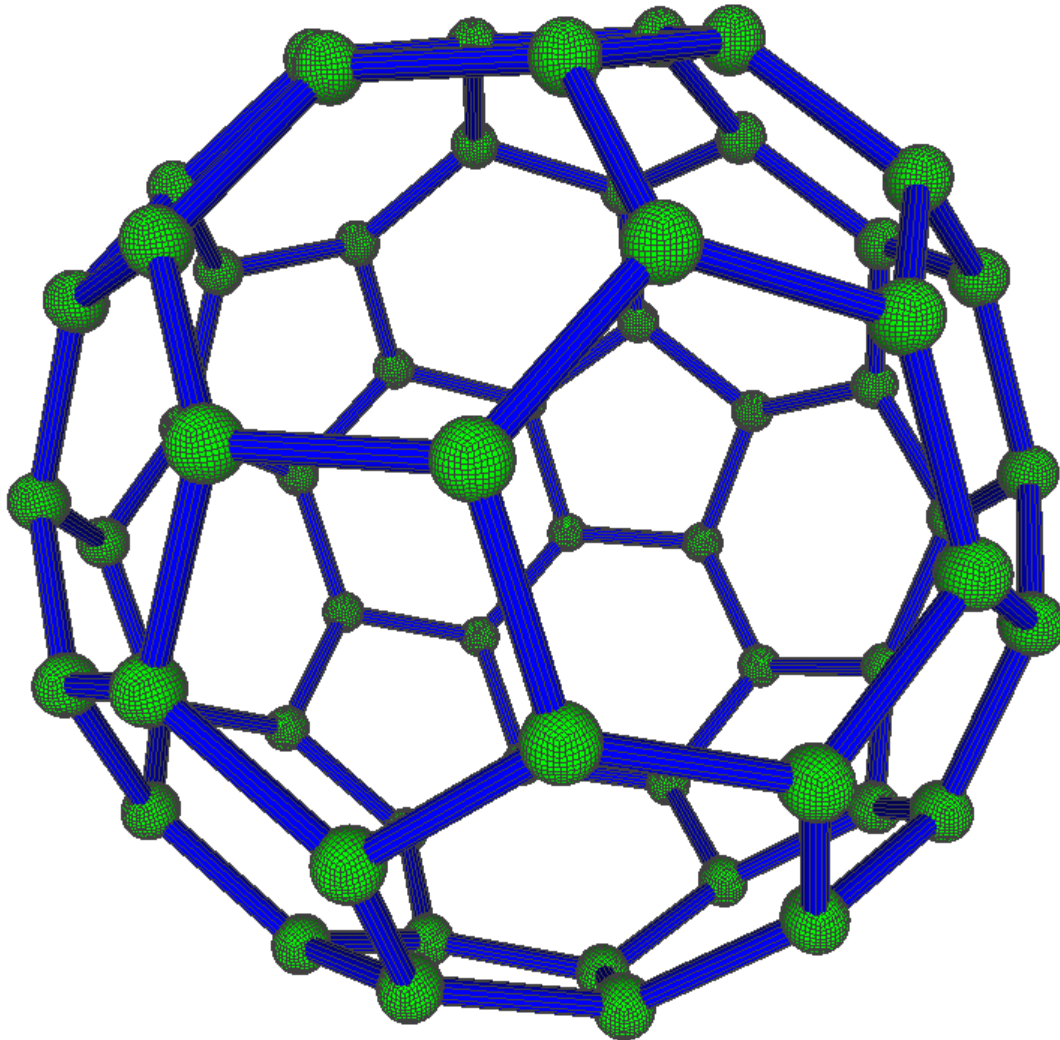
example5.off



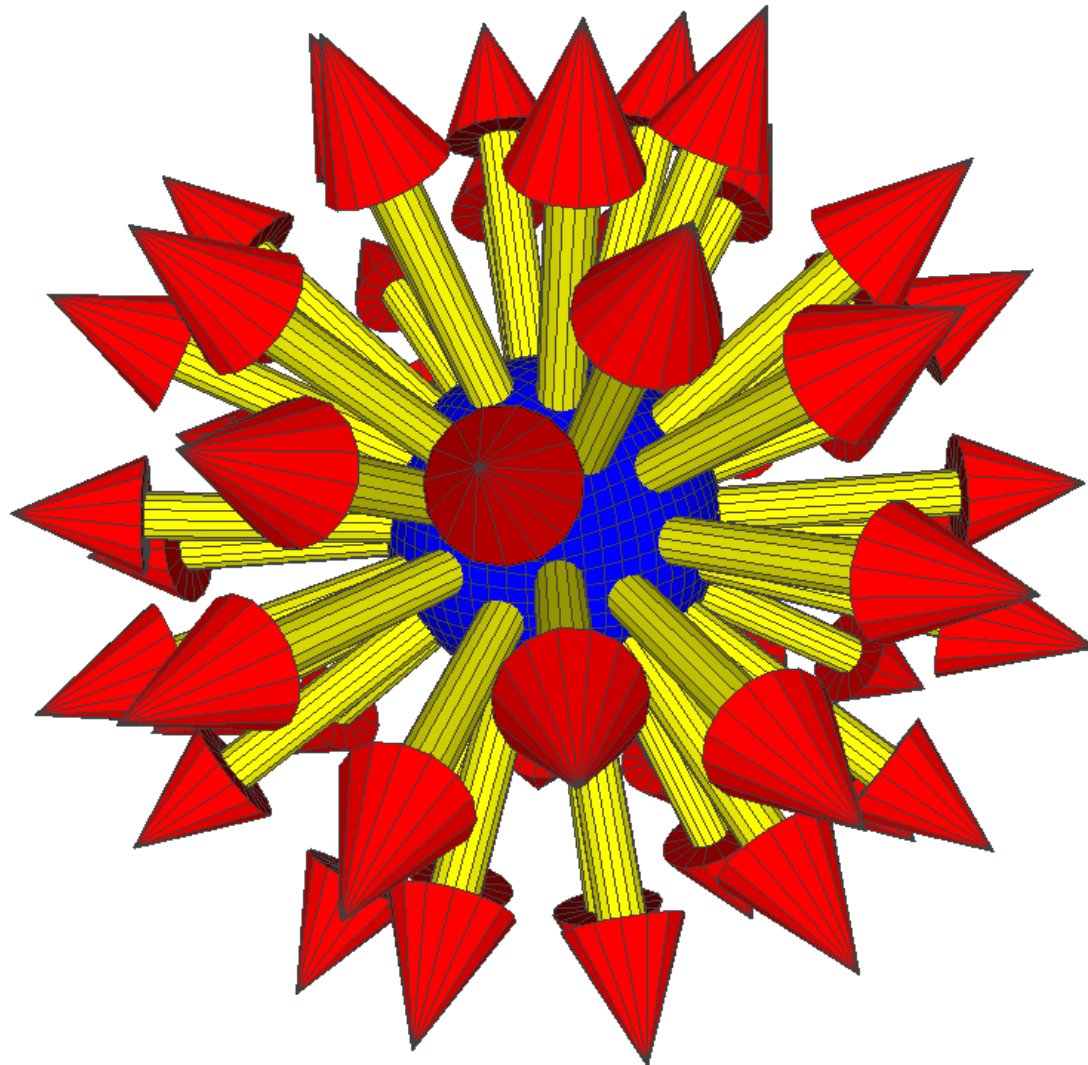
example6.off



example7.off



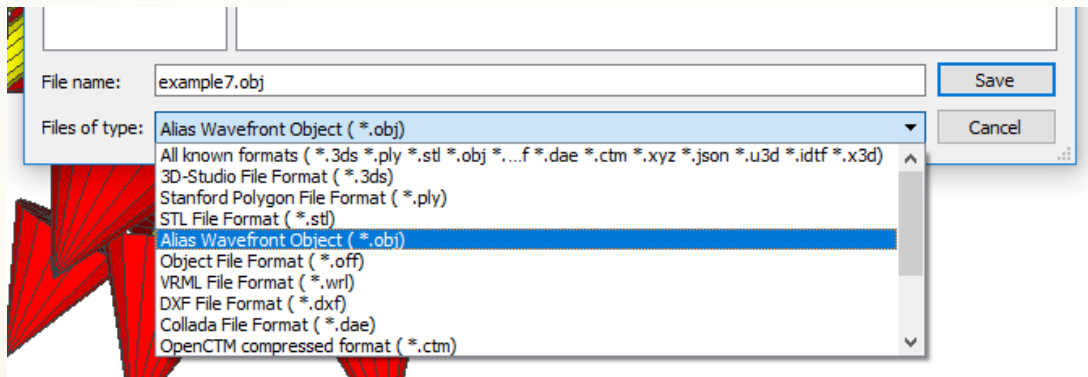
example8.off



3D modelio viešinimas

<https://sketchfab.com> svetainėje

- Naudojant *MeshLab* programą 3D modelį reikia konvertuoti iš OFF formato į OBJ formatą:
 - 1) File → Export Mesh As...
 - 2) Pasirinkti *.obj formatą:

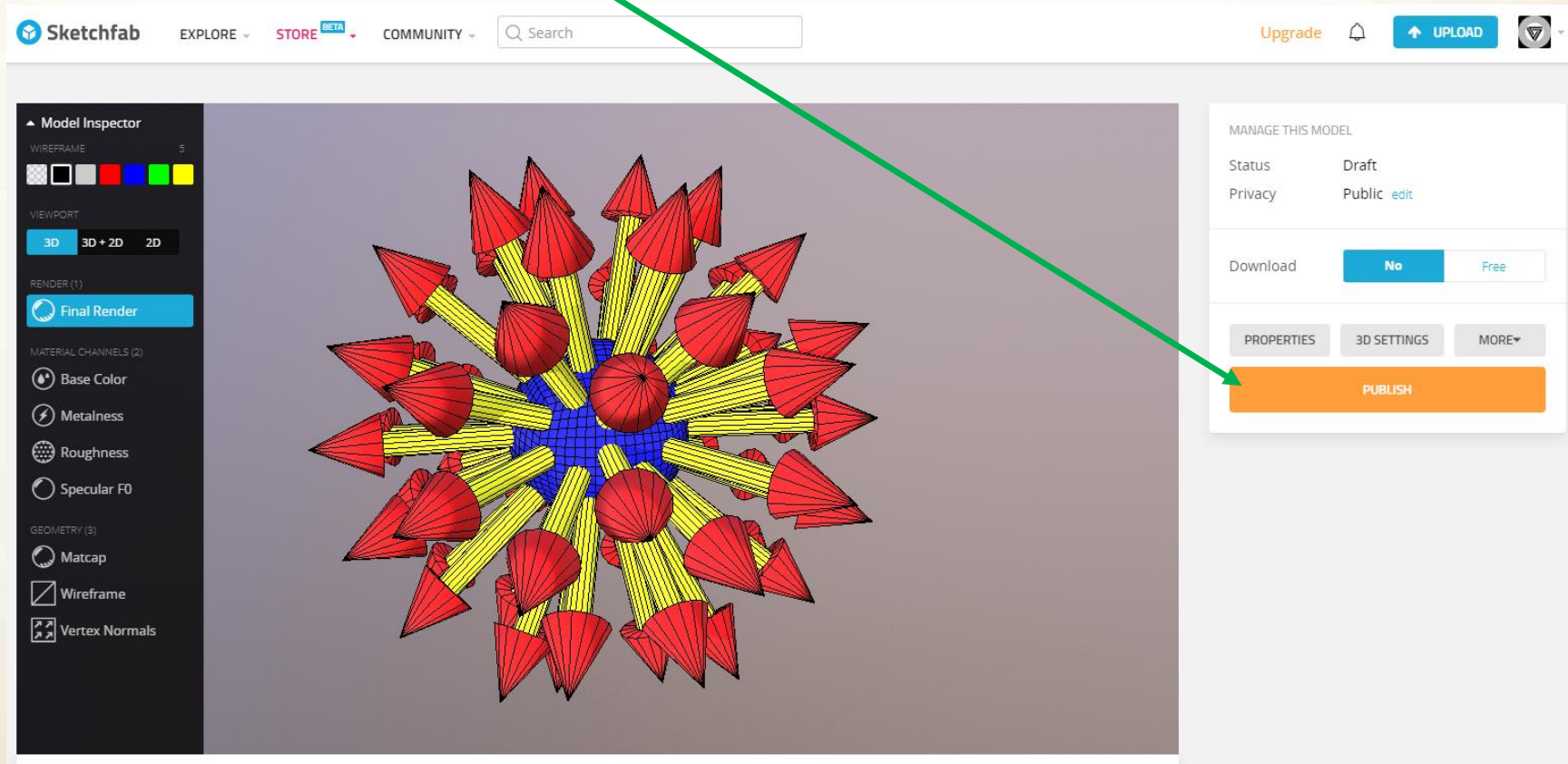


- 3) Išsaugojus bus gauti 2 failai: *.obj ir *.mtl.
- 4) Abu šiuos failus reikia įtraukti į 7z archyvą.
- 5) Archyvą įkelti į [sketchfab](https://sketchfab.com) sistemą, susikūrus nemokamą paskyrą.

3D modelio viešinimas

<https://sketchfab.com> svetainėje

Nustačius režimą „Public“, 3D modelis tampa viešai prieinamas, juo galima dalintis nuoroda.



Ačiū už dėmesį.