

Parser de obiecte wavefront (.obj) în format JSON

Autoevaluare Proiect

Bartolomei Vlad, grupa 30239

1. Contextul problemei

Într-o lume în care apar tot mai multe tehnologii și tot atâtea formate de fișiere, este important ca unele fișiere să poată fi transferate ușor. Pentru aceasta, formatele *xml* și *json* sunt formate de fișiere universale, ușor de înțeles și frecvent folosite pentru astfel de transferuri.

2. Cerință (textul problemei)

- Să se proiecteze un parser care ia un fișier de tip [wavefront \(.obj\)](#) obligatoriu triangular¹ (*triangulate mesh* bifat la export) într-un fișier format [JSON](#).
- Pentru acest proiect ne interesează doar vârfurile și fețele. Astfel, ne vor interesa doar liniile care încep cu **v** și cu **f**.
- Se va face abstracție de textură sau normale la suprafață aferente fiecărui punct.

```
# List of geometric vertices, with (x, y, z, [w]) coordinates, w is optional and defaults to 1.0.
v 0.123 0.234 0.345 1.0
v ...
...
# List of texture coordinates, in (u, [v, w]) coordinates, these will vary between 0 and 1. v, w are
optional and default to 0.
vt 0.500 1 [0]
vt ...
...
# List of vertex normals in (x,y,z) form; normals might not be unit vectors.
vn 0.707 0.000 0.707
vn ...
...
# Parameter space vertices in (u, [v, w]) form; free form geometry statement (see below)
vp 0.310000 3.210000 2.100000
vp ...
...
# Polygonal face element (see below)
f 1 2 3
f 3/1 4/2 5/3
f 6/4/1 3/5/3 7/6/5
f 7//1 8//2 9//3
f ...
...
# Line element (see below)
l 5 8 1 2 4 9
```

¹ În Computer Vision, un obiect poate avea și fețe determinate de 4 sau mai multe puncte. Un minim absolut pentru o față este de 3 puncte (triunghi).

3. Complexitate

Baza aplicației include parsarea unui fișier .obj (triangular mesh) și transformarea acestuia într-un format JSON. Aceasta implică:

- Recunoașterea și extragerea coordonatelor vârfurilor (liniile care încep cu **v**).
- Recunoașterea și extragerea informațiilor despre fețe (liniile care încep cu **f**).
- Ignorarea altor tipuri de informații prezente în fișierul .obj (de exemplu, texturi și normale).

Comparativ cu alte softuri existente, aplicația se concentrează pe o subset de date relevante, simplificând astfel procesul.

URL exemplu similar: <https://github.com/charalambos/OBJLoader/tree/master>

3.1. V for Vertex

Formatul general al unei linii ce definește un punct (vertex) este:

```
v x_float y_float z_float
```

V indică linia corespunzătoare definirii unui vertex.

X_float este un număr real ce amplasează punctul pe axa Ox în World coordinate system.

Y_float - idem, pentru axa Oy

Z_float - idem, pentru axa Oz

3.2. F for Face

Formatul general al unei linii ce definește o față este:

```
f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3
```

F indică linia corespunzătoare definirii unei fețe. O față este mereu în formă de triunghi, fiind definită astfel de trei puncte.

O tripletă de tipul **v/vt/vn** (vertex, vertex texture, vertex normal) face referire la liniile de vertices (cele expuse la 3.1.) definite anterior în document și definește pentru fiecare vertex asociat:

- Numărul vertex-ului referit din document
- Numărul vertex-texture-ului referit (liniile cu vt; nu sunt de interes în acest proiect)
- Numărul normalei vertex-ului referite (liniile cu vn; idem)

Proiectantul trebuie să știe că o față este determinată de tripleta (v1, v2, v3).

4.Complexitate avansată Lex & Yacc

- **Reguli pentru Vertex și Face:** Definirea regulilor specifice pentru identificarea și procesarea liniilor **v** și **f**.
- **Gestionarea float și int:** Lexer-ul recunoaște și returnează valori de tip float și integer.
- **Gestionarea caracterului /:** Lexer-ul recunoaște acest caracter necesar în triplete de tipul INTEGER/INTEGER/INTEGER, întrucât nu întotdeauna caracterul spațiu servește ca delimitator pentru acest tip de fișiere
- **Reguli pentru altele:** Gestionarea liniilor neinteresante pentru parser, prin ignorarea acestora (la întâlnirea lor, lexer-ul nu face nicio acțiune).
- **Structura arborelui de parsare:** Utilizarea unui arbore de parsare pentru a organiza structurile Vertex și Face.

Itemi originali:

- Reguli pentru **VERTEX_LINE** și **FACE_LINE** în Yacc.
- Reguli pentru **OTHERS** și **TERMINATOR** pentru flexibilitatea și robustețea parserului.
- Gestionarea tripletelor **v/vt/vn** în **FACE_DATA**.
- Scrierea coordonatelor și fețelor în format JSON.
- Gestionarea erorilor și terminarea liniei.

4.1. Recunoașterea vertex-urilor

```
// Coordonatele unui Vertex pe axa 0x, 0y, 0z sunt retinute in aceasta structura
typedef struct Vertex {
    float x, y, z;
} Vertex;
```

```
L    : VERTEX VERTEX_LINE TERMINATOR
      | FACE FACE_LINE TERMINATOR
      | OTHERS TERMINATOR
      | TERMINATOR
      | /*empty*/
      ;
```

```
VERTEX_LINE: FLOAT FLOAT FLOAT {
    // se ajunge la o linie de tipul "v 1.670000 2.340000 -5.400000"
    vertices[noOfVertices].x = $1.fval;
    vertices[noOfVertices].y = $2.fval;
    vertices[noOfVertices].z = $3.fval;
    noOfVertices++;
}
```

4.2. Recunoașterea fețelor

```
// Face este o tripletă de tipul "v/vt/vn" (vertex/vertex texture/vertex normal)
// v = v1, vt = v2, vn = v3
typedef struct Face {
    int v1, v2, v3;
} Face;
```

```
L : VERTEX VERTEX_LINE TERMINATOR
  | FACE FACE_LINE TERMINATOR
  | OTHERS TERMINATOR
  | TERMINATOR
  | /*empty*/
  ;
```

```
FACE_LINE: FACE_DATA FACE_DATA FACE_DATA
          ;

FACE_DATA: INTEGER '/' INTEGER '/' INTEGER {
            // se ajunge la o tripletă de tipul "v/vt/vn"
            faces[noOfFaces].v1 = $1.ival;
            faces[noOfFaces].v2 = $2.ival;
            faces[noOfFaces].v3 = $3.ival;
            noOfFaces++;
          }
          ;
```

4.3. Alte mențiuni

- Tokenii **FACE** și **VERTEX** sunt responsabili pentru recunoașterea caracterelor **f**, respectiv **v**

```
Vertex vertices[5000];
Face faces[5000];
int noOfVertices = 0;
int noOfFaces = 0;
```

- Parsarea în JSON a necesitat memorarea tuturor acestor rezultate în structurile create mai sus (vertices[] și faces[]), iar în main parsarea a avut loc facil, respectând formatul unui fișier json:

```
printf("{\n\t\"vertices\": [\n");
for(int i = 0; i < noOfVertices; i++)
{
    printf("\t\t[%f, %f, %f]", vertices[i].x, vertices[i].y, vertices[i].z);
    if(i < noOfVertices - 1)
        printf(", ");
    printf("\n");
}
printf("\t],\n");
```

Se va obține ceva de genul:

```
{
    "Vertices": [
        [v1, v2, v3],
        ...
        [vn-2, vn-1, vn]
    ],
```

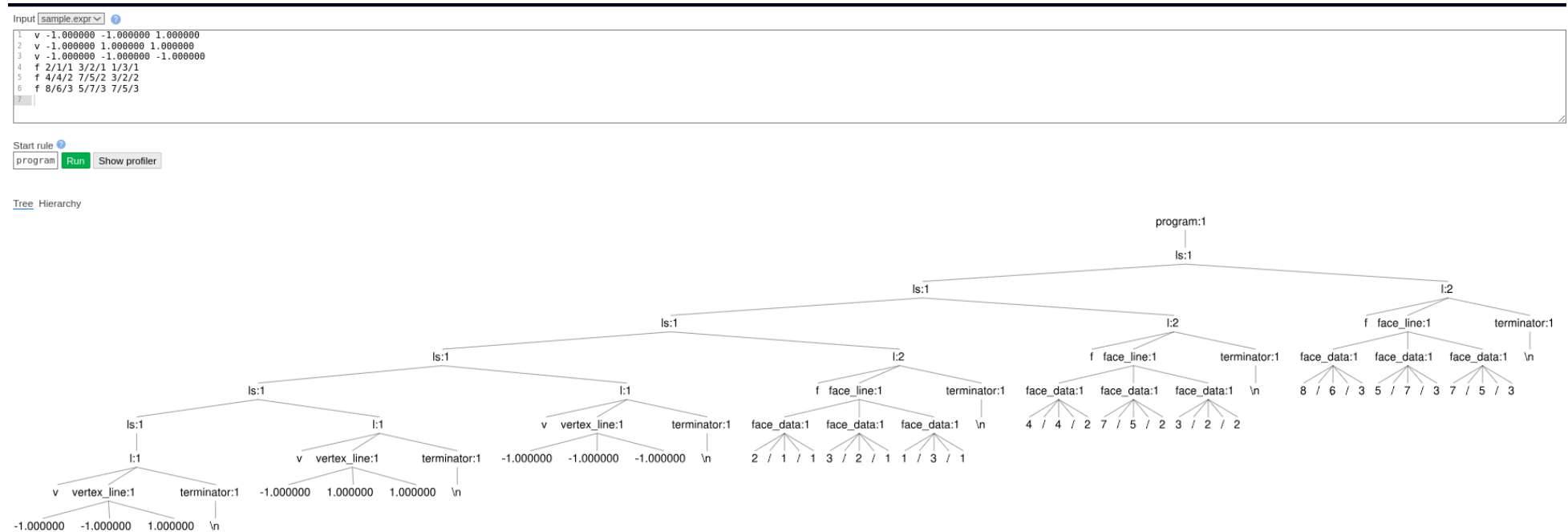
```
printf("\t\"faces\": [\n");
for(int i = 0; i < noOfFaces-2; i = i + 3)
{
    printf("\t\t[%d, %d, %d]",
        faces[i].v1 - 1,
        faces[i+1].v1 - 1,
        faces[i+2].v1 - 1);
    if(i < noOfFaces - 4)
        printf(", ");
    printf("\n");
}
printf("\t]\n");
```

Această secvență parcurge faces, urmând a se obține ceva de genul:

```
"Faces": [
    [f1, f2, f3],
    ...
```

$$[f_{m-2}, f_{m-1}, f_m]$$

5. Adâncimea arborelui de parsare



6. Rezolvare ambiguitate

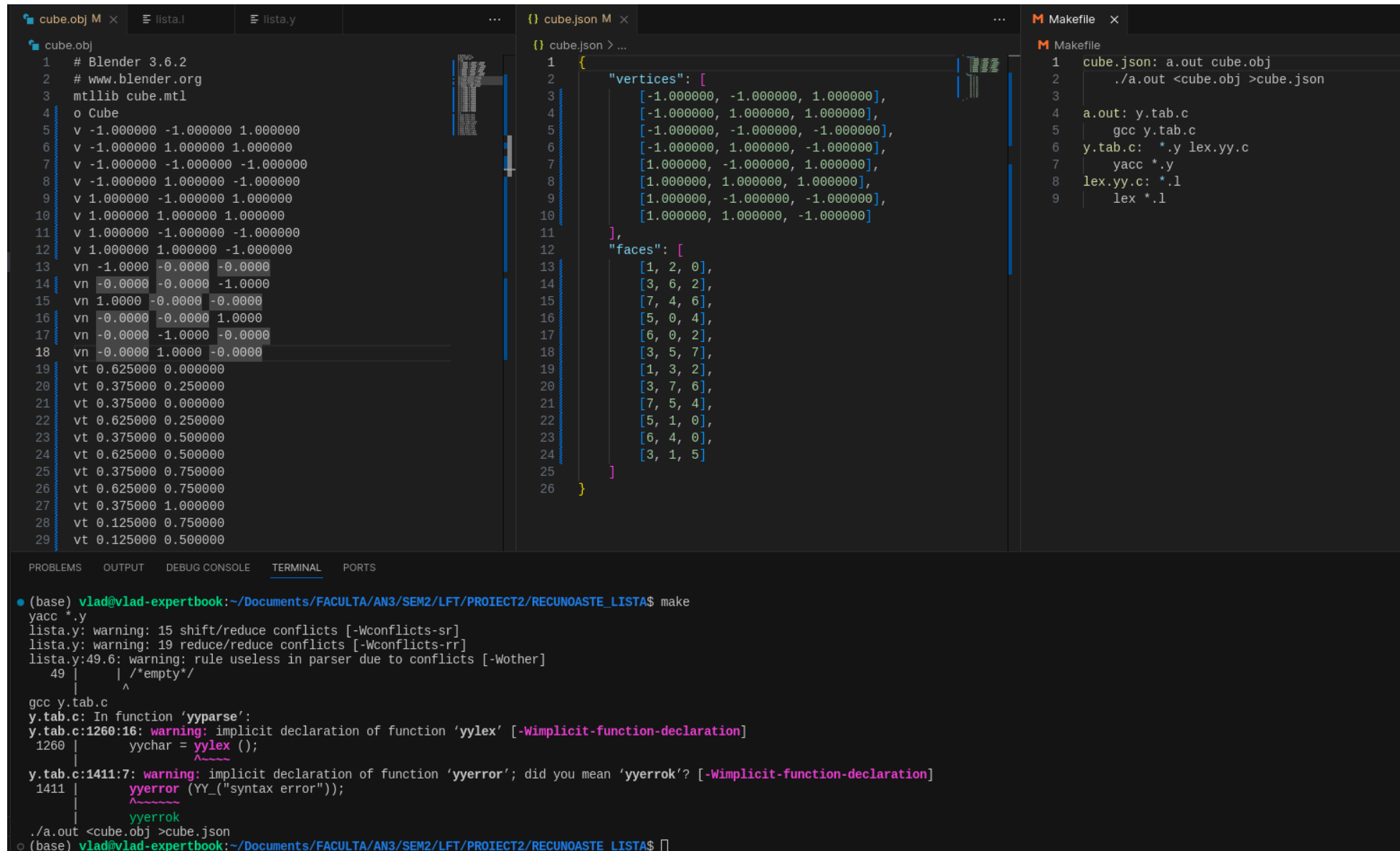
Ambiguitățile sunt rezolvate prin:

- Definirea regulilor clare pentru `VERTEX_LINE` și `FACE_LINE`.
- Utilizarea regulilor `TERMINATOR` pentru a gestiona terminarea liniei.
- Tratarea altor tipuri de date folosind regula `OTHERS`, prevenind astfel confuziile.

7. Avantajele utilizării gramaticii alese. Flexibilitate în compunerea simbolurilor descrise

- **Flexibilitate:** Permite gestionarea și ignorarea liniilor neinteresante.
- **Claritate:** Reguli clare pentru liniile de tip vertex și face.
- **Comparare:** Comparativ cu un exemplu similar, această abordare oferă o structură mai clară și modulară, îmbunătățind întreținerea și extensibilitatea codului.

8. Scenariu de test. Testare și validare



```
cube.obj M x  llsta.l  llsta.y  ...  {} cube.json M x  ...  M Makefile x
cube.obj
1 # Blender 3.6.2
2 # www.blender.org
3 mtl lib cube.mtl
4 o Cube
5 v -1.000000 -1.000000 1.000000
6 v -1.000000 1.000000 1.000000
7 v -1.000000 -1.000000 -1.000000
8 v -1.000000 1.000000 -1.000000
9 v 1.000000 -1.000000 1.000000
10 v 1.000000 1.000000 1.000000
11 v 1.000000 -1.000000 -1.000000
12 v 1.000000 1.000000 -1.000000
13 vn -1.0000 -0.0000 -0.0000
14 vn -0.0000 -0.0000 -1.0000
15 vn 1.0000 -0.0000 -0.0000
16 vn -0.0000 -0.0000 1.0000
17 vn -0.0000 -1.0000 -0.0000
18 vn -0.0000 1.0000 -0.0000
19 vt 0.625000 0.000000
20 vt 0.375000 0.250000
21 vt 0.375000 0.000000
22 vt 0.625000 0.250000
23 vt 0.375000 0.500000
24 vt 0.625000 0.500000
25 vt 0.375000 0.750000
26 vt 0.625000 0.750000
27 vt 0.375000 1.000000
28 vt 0.125000 0.750000
29 vt 0.125000 0.500000

cube.json M x  ...
{} cube.json > ...
1 {
2   "vertices": [
3     [-1.000000, -1.000000, 1.000000],
4     [-1.000000, 1.000000, 1.000000],
5     [-1.000000, -1.000000, -1.000000],
6     [-1.000000, 1.000000, -1.000000],
7     [1.000000, -1.000000, 1.000000],
8     [1.000000, 1.000000, 1.000000],
9     [1.000000, -1.000000, -1.000000],
10    [1.000000, 1.000000, -1.000000]
11  ],
12   "faces": [
13    [1, 2, 0],
14    [3, 6, 2],
15    [7, 4, 6],
16    [5, 0, 4],
17    [6, 0, 2],
18    [3, 5, 7],
19    [1, 3, 2],
20    [3, 7, 6],
21    [7, 5, 4],
22    [5, 1, 0],
23    [6, 4, 0],
24    [3, 1, 5]
25  ]
26 }

Makefile x
M Makefile
1 cube.json: a.out cube.obj
2   ./a.out <cube.obj >cube.json
3
4 a.out: y.tab.c
5   gcc y.tab.c
6 y.tab.c: *.y lex.yy.c
7   yacc *.y
8 lex.yy.c: *.l
9   lex *.l

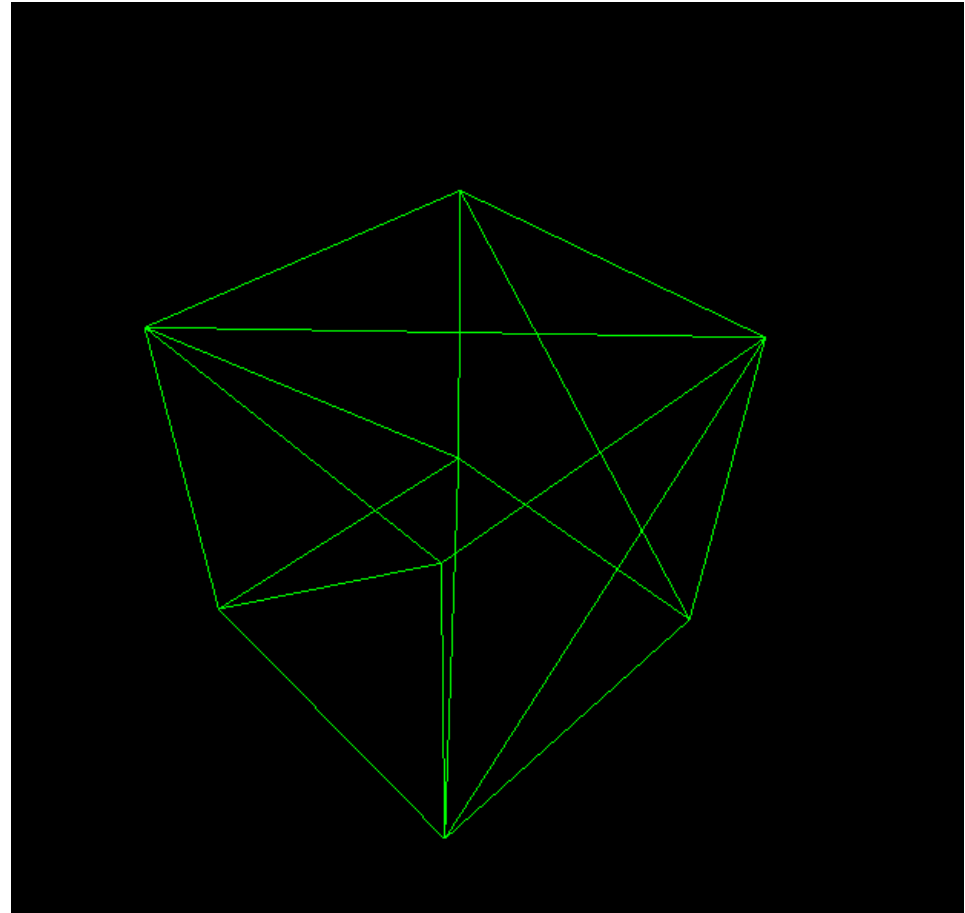
(base) vlad@vlad-expertbook:~/Documents/FACULTA/AN3/SEM2/LFT/PROIECT2/RECUNOASTE_LISTA$ make
yacc *.y
lista.y: warning: 15 shift/reduce conflicts [-Wconflicts-sr]
lista.y: warning: 19 reduce/reduce conflicts [-Wconflicts-rr]
lista.y:49:6: warning: rule useless in parser due to conflicts [-Wother]
49 | | /*empty*/
   | ^
gcc y.tab.c
y.tab.c: In function 'yyparse':
y.tab.c:1260:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
1260 |     yychar = yylex ();
      |                ^~~~~
y.tab.c:1411:7: warning: implicit declaration of function 'yyerror'; did you mean 'yyerrok'? [-Wimplicit-function-declaration]
1411 |     yyerror (YY_("syntax error"));
      |     ^~~~~~
      |     yyerrok
./a.out <cube.obj >cube.json
(base) vlad@vlad-expertbook:~/Documents/FACULTA/AN3/SEM2/LFT/PROIECT2/RECUNOASTE_LISTA$
```

Unde cube.obj reprezintă reprezentarea unui cub în format .obj:

```

cube.obj M x llista.l llista.y
cube.obj
1 # Blender 3.6.2
2 # www.blender.org
3 mtllib cube.mtl
4 o Cube
5 v -1.000000 -1.000000 1.000000
6 v -1.000000 1.000000 1.000000
7 v -1.000000 -1.000000 -1.000000
8 v -1.000000 1.000000 -1.000000
9 v 1.000000 -1.000000 1.000000
10 v 1.000000 1.000000 1.000000
11 v 1.000000 -1.000000 -1.000000
12 v 1.000000 1.000000 -1.000000
13 vn -1.0000 -0.0000 -0.0000
14 vn -0.0000 -0.0000 -1.0000
15 vn 1.0000 -0.0000 -0.0000
16 vn -0.0000 -0.0000 1.0000
17 vn -0.0000 -1.0000 -0.0000
18 vn -0.0000 1.0000 -0.0000
19 vt 0.625000 0.000000
20 vt 0.375000 0.250000
21 vt 0.375000 0.000000
22 vt 0.625000 0.250000
23 vt 0.375000 0.500000
24 vt 0.625000 0.500000
25 vt 0.375000 0.750000
26 vt 0.625000 0.750000
27 vt 0.375000 1.000000
28 vt 0.125000 0.750000
29 vt 0.125000 0.500000
30 vt 0.875000 0.500000
31 vt 0.625000 1.000000
32 vt 0.875000 0.750000
33 s 0
34 f 2/1/1 3/2/1 1/3/1
35 f 4/4/2 7/5/2 3/2/2
36 f 8/6/3 5/7/3 7/5/3
37 f 6/8/4 1/9/4 5/7/4
38 f 7/5/5 1/10/5 3/11/5
39 f 4/12/6 6/8/6 8/6/6
40 f 2/1/1 4/4/1 3/2/1
41 f 4/4/2 8/6/2 7/5/2
42 f 8/6/3 6/8/3 5/7/3
43 f 6/8/4 2/13/4 1/9/4
44 f 7/5/5 5/7/5 1/10/5
45 f 4/12/6 2/14/6 6/8/6

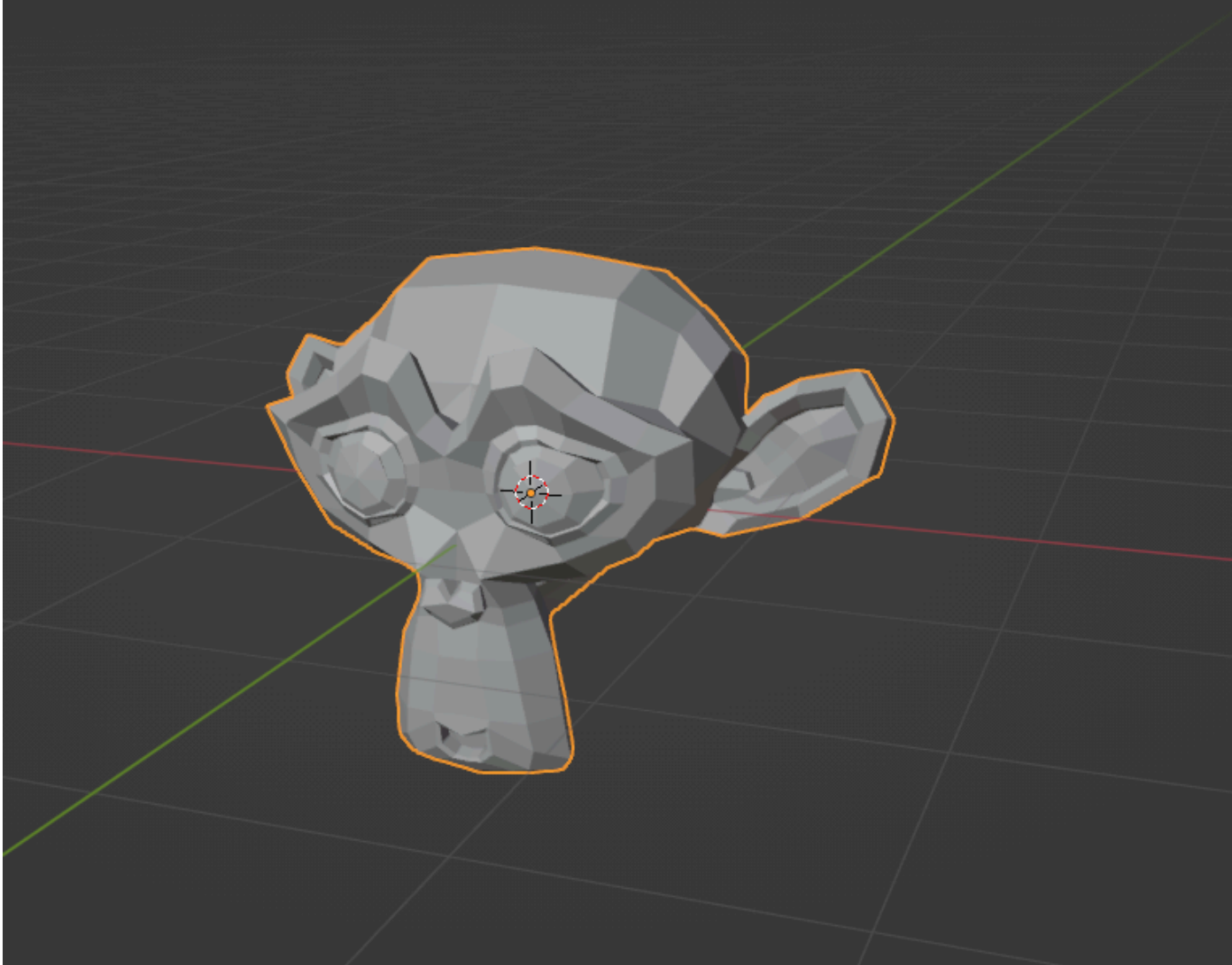
```

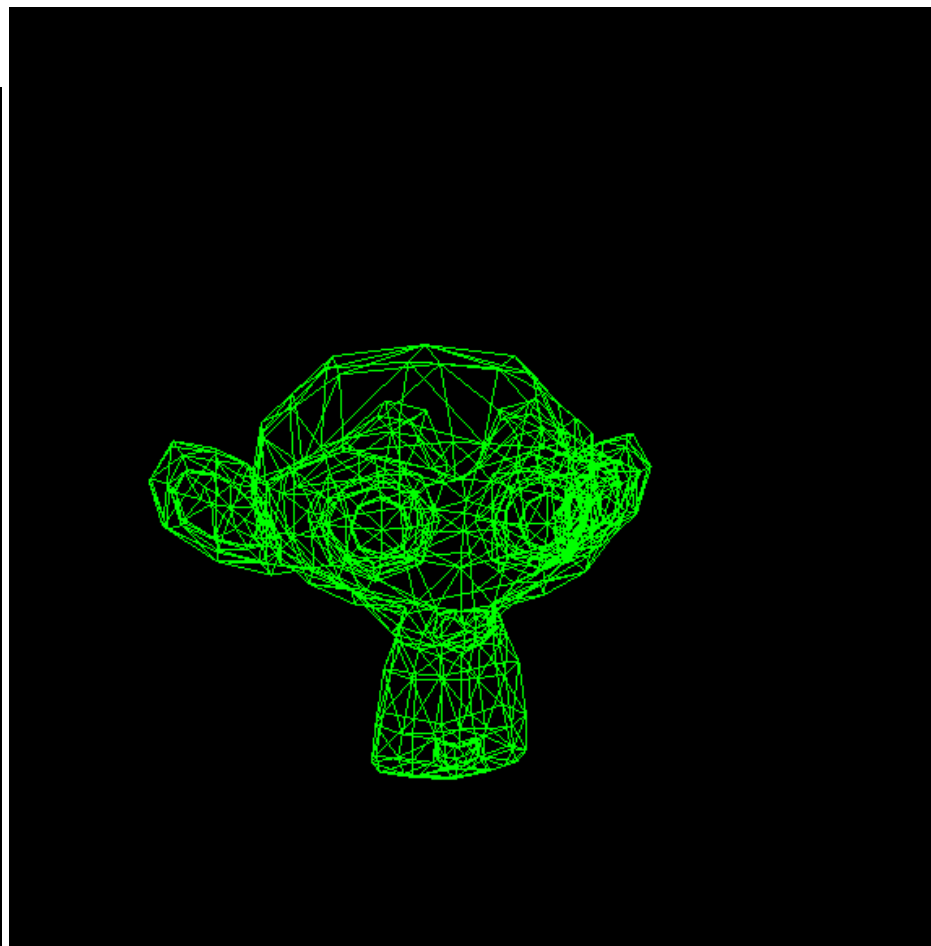
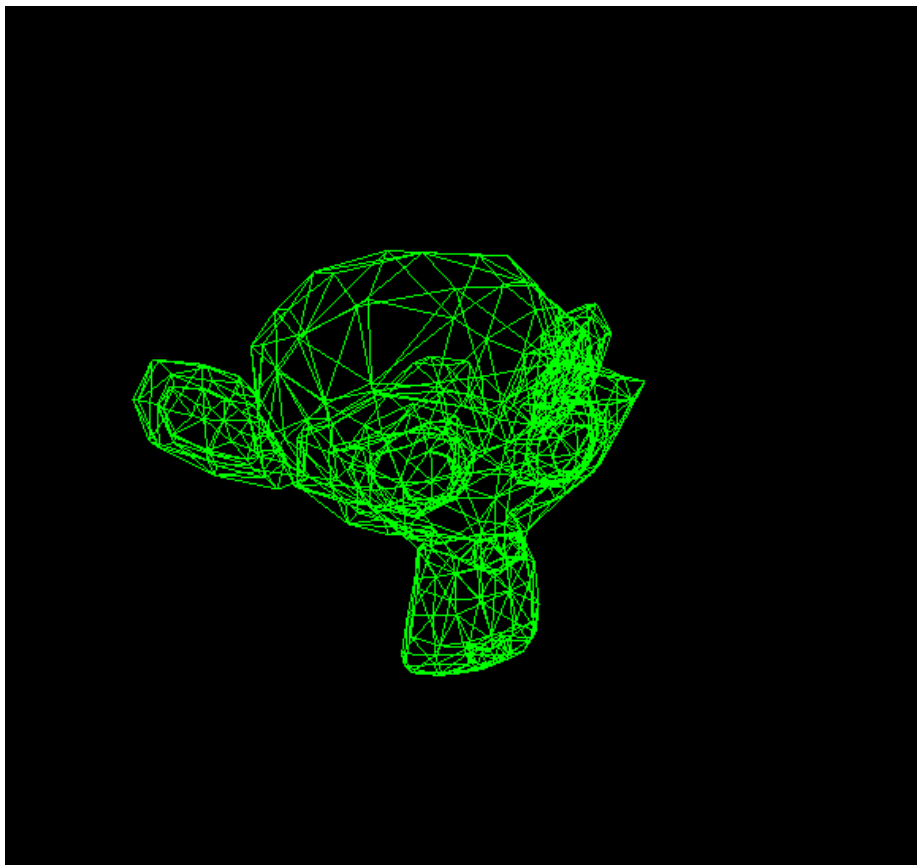


Validarea (vizualizarea) rezultatului poate fi făcută cu ajutorul unui mic program *three.js* realizat de mine care interpretează un fișier JSON cu specificațiile cerute și afișează obiectul.

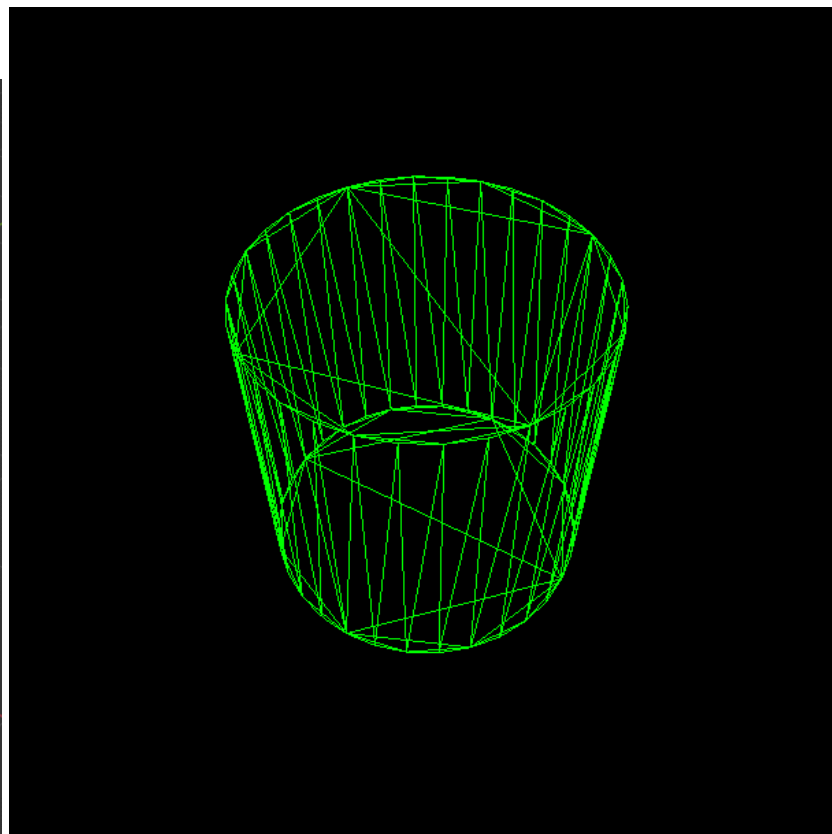
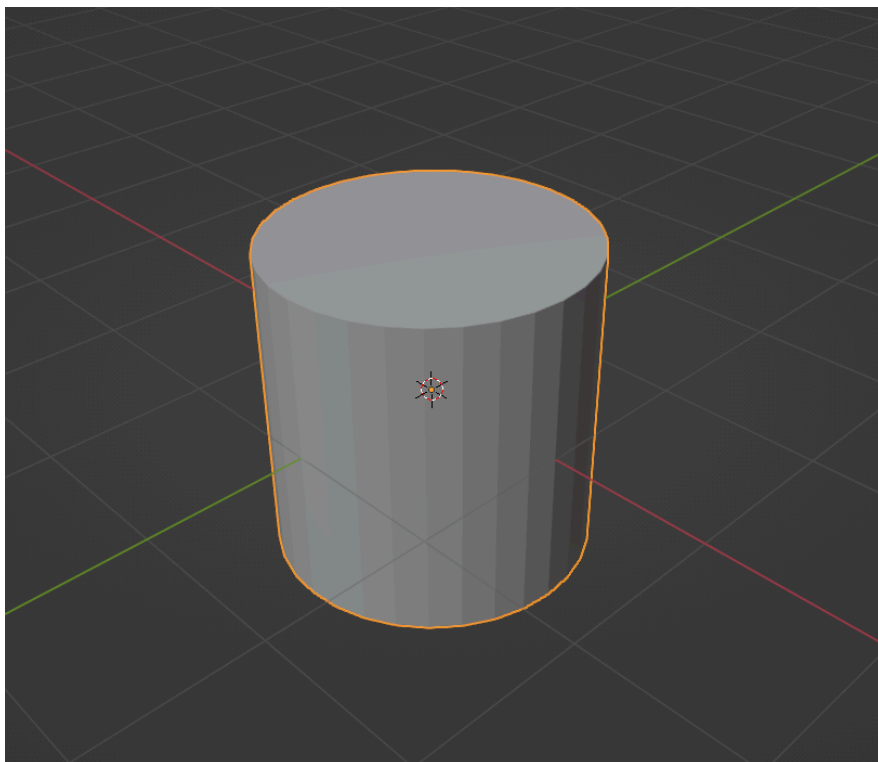
9. Scenarii de test suplimentar

- Pentru obiectul Suzanne (mănușă standard din blender)
- Fișierul .obj aferent și .json generat se găsesc în arhiva **Test_Scenario_Suzanne.zip**





- Pentru obiectul Cylinder (un cilindru standard din blender)
 - Fișierul .obj aferent și .json generat se găsesc în arhiva **Test_Scenario_Cylinder.zip**



10. Concluzii și îmbunătățiri propuse [3 propuneri]

În procesul de dezvoltare a acestui parser, în timpul detecției fețelor am salvat intenționat și informațiile despre vt și vn. Prin neignorarea acestora, am dat posibilitatea extinderii utilității parserului astfel:

- a. Recunoașterea liniilor texturii
- b. Recunoașterea liniilor normalelor vertex-urilor
- c. Includerea acestora în fișierul final JSON
- d. Într-un fișier .obj va exista o linie care începe cu **mtllib** și va duce către *.mtl, în care va fi definită textura obiectului. Aplicația ar putea suporta în viitor și parsarea acestei texturi
- e. Dezvoltarea aplicației de test cu o parte de client (în care încarci un obiect .obj) și una de server (care face toate rulările în spate, folosindu-se de comanda **make**).