

**UNIVERSITATEA TEHNICĂ A MOLDOVEI**

**FACULTATEA CALCULATOARE, INFORMATICA,  
MICROELECTRONICA**

**GRAFICA PE CALCULATOR**

**ÎNDRUMĂR METODIC PENTRU LUCRĂRI DE LABORATOR**  
**(draft)**

**2022**

**Chișinău 2022**

## Introducerea în biblioteca grafică p5.js

### Ce este p5.js?

P5.js este o bibliotecă scrisă în limbajul de programare JavaScript, utilizată pentru crearea și vizualizarea imaginilor interactive cu ajutorul primitivelor grafice simple. P5.js permite crearea graficii pe calculator folosind un limbaj de programare. Această permite integrarea simplă a codurilor scrise în pagini web prin adăugarea codului scris încu-un document HTML.

P5.js este gratis, open-sources și independent de platformă, deci aplicațiile pot fi rulate pe orice sistem de operare, deasemenea p5.js are o familie mare de limbaje și medii programare înrudite, aceste sunt prezentate în figura 1.

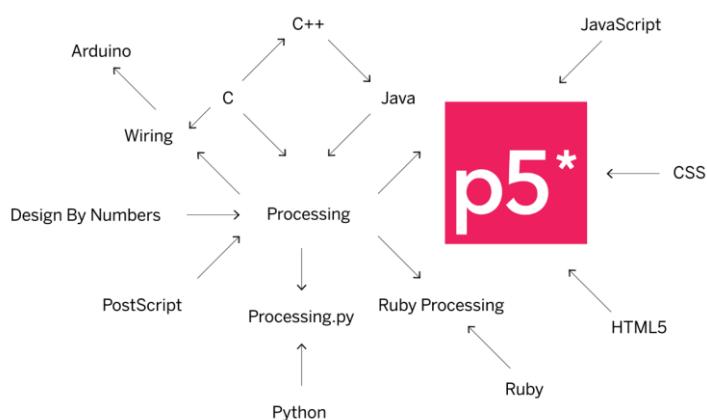


Figura 1. Limbaje și medii de programare înrudite cu p5.js

### Creare unui program simplu în P5.js

Înnainte de a începe crearea programelor priprii trebuie să știm că orice figură creată în mediul p5.js este legată de sistemul de coordonate, originea sistemului de coordonate în orice program este colțul din stânga sus al ecranului. Axa verticală se numește axa Y, iar cea orizontală axa X. Creșterea valorilor pentru coordonatele x și y sunt prezentate în figura 2.

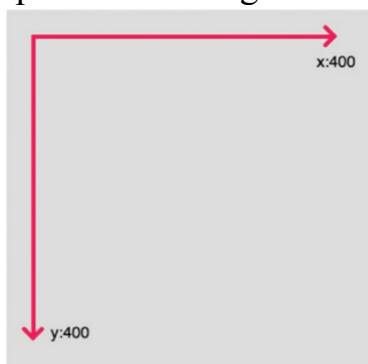


Figura 2. Sistemul de coordinate în mediul p5.js

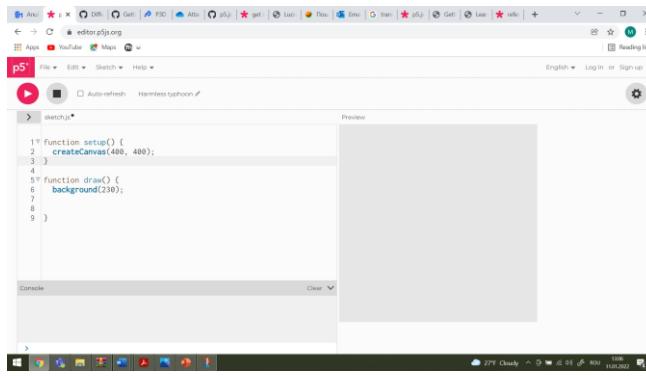
Pentru crearea unui program simplu în p5.js trebuie să utilizăm următorul şablon:

```

function setup(){
    createCanvas(400,400);
}
function draw(){
    background(220);
}

```

Rezultatul programului și exemplul editorului online p5.js este arătat în figura 3.



**Figura 3. Fereastră de lucru în mediul p5.js**

**Funcția setup()** – se apelează o singură dată la începutul programului. Este folosită pentru a setarea proprietăților inițiale a mediului de lucru, cum ar fi dimensiunea și culoarea ecranului, precum și pentru încărcarea fișiere multimedia la lansarea programului, cum ar fi imagini și fonturi. Poate exista o singură funcție setup() per program și nu ar trebui apelată după execuția inițială.

**Notă:** Variabilele declarate în setup() nu sunt disponibile în alte funcții, inclusiv draw().

#### **Exemplu:**

```

function setup() {
  createCanvas();
}
  
```

createCanvas – crează un element canvas (pînză) și setază dimensiunea acestuia în pixeli. Această metodă ar trebui apelată o singură dată la începutul programului. Apelarea createCanvas de mai multe ori într-un cod va avea ca rezultat un comportament foarte imprevizibil. Dacă doriți mai mult de o pânză de desen, puteți utiliza createGraphics.

Sintaxă metodei este următoarea:

`createCanvas(w, h, [renderer])`

unde:

w - număr: lățimea pînzei;

h - număr: înălțimea pînzei;

constanta renderului: P2D - dacă originea sistemului de coordonate este în colțul stâng sus a ecranului sau WEBGL – dacă originea sistemului de coordonate este în centrul pînzei este specific pentru grafica 3D (optională).

Dacă createCanvas () nu este utilizat în program pînzei o să-i fie atribuită valoarea implicită 100x100.

**Funcția draw()** se apelează imediat după setup(), execută în continuu rîndurile de cod care sunt incluse în corpul său, pînă la svârșitul programului sau pînă la apelarea noLoop().

**Notă:** dacă în setup() este apelată funcția noLoop(), funcția draw() se va executa o singură dată.

Sintaxa funcției este:

```
function draw() {  
-----  
}
```

**Funcția background()** setează culoarea utilizată ca fundal canvas-ului. Implicit fundalul este transparent. Această funcție de obicei se utilizează în draw() pentru a șterge fereastra de afișare la începutul fiecărui cadru dar, poate fi utilizată și în interiorul funcției setup(), pentru a seta fundalul pe primul cadru al animației sau dacă fundalul trebuie setat o singură dată.

Culoarea este specificată în RGB, HSB sau HSL, în dependență de colorMode. (Implicit modul este - RGB, deci fiecare valoare este în diapazonul 0 ÷ 255).

Sintaxa funcției este:

```
background(color)  
background(colorstring, [a])  
background(gray, [a])  
background(v1, v2, v3, [a])  
background(values)  
background(image, [a])
```

color:

orice valoare creată cu ajutorul funcției color();

colorstring String:

un string (denumirea culorii în engleză), formatele posibile: un număr întreg rgb() sau rgba(), procent rgb() sau rgba(), 3-cifre hexazecimale, 6-cefre hexazecimale;

a (număr):

opacitatea fundalului în raport cu gama de culori curentă (implicit 0-255) (obișnuit);

gray (număr):

specifică valoarea între alb și negru;

v1 (număr)

specifică valoarea roșie sau valoarea nuanței (în dependență de gama curentă de culori);

v2 (număr):

specifică valoarea verde sau valoarea saturăției (în dependență de gama curentă de culori)

v3 (număr):

specifică valoarea albastră sau valoarea luminozității (în dependență de gama curentă de culori)

values Number []:

un masiv care, conține componentele roșie, verde, albastră și alfa a culorilor;

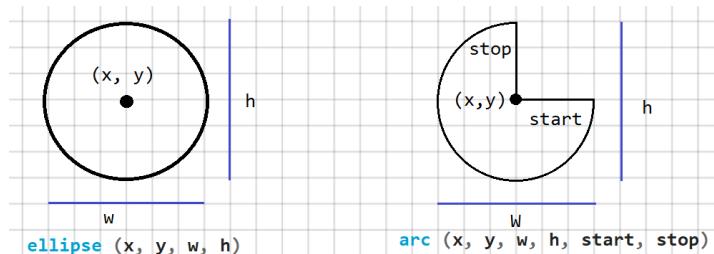
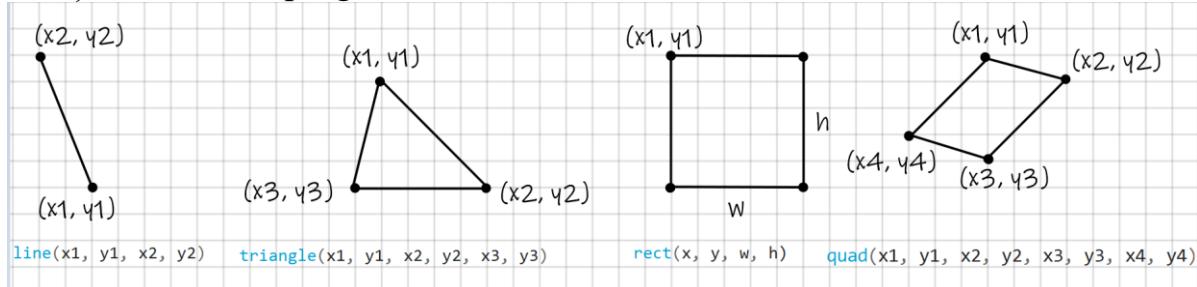
image:

imaginile create utilizând loadImage() sau createImage(), pentru a fi setată ca fon;

## Capitolul I

### 1.1 Crearea primitivelor grafice 2D simple

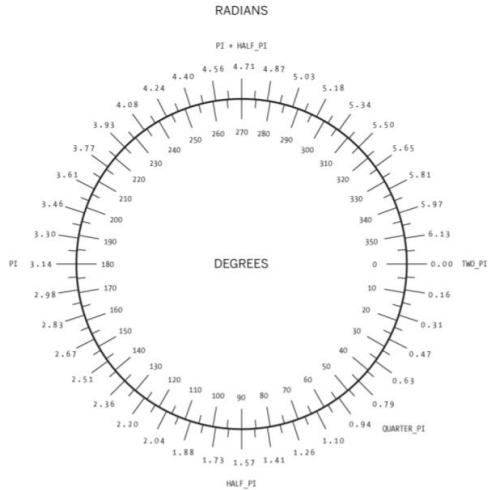
Primitivele grafice simple reprezintă figurile geometrice care pot fi create cu ajutorul funcțiilor din biblioteca grafică P5.js. Cele mai simple primitive grafice sunt primitivele grafice 2D, în figura 1.1 arătată corespunderea punctelor figurilor geometrice și parametrilor care trebuie indicate ca argumentele funcțiilor în codul programului.



**Figura 1.1. Relația dintre punctele geometrice și parametrii funcțiilor P5.js**

**Funcția arc ():** desenează un arc pe ecran. Dacă este apelat doar cu x, y, w, h, start și stop, arcul va fi desenat și umplut ca un segment de cerc deschis. Dacă este specificat parametru *mod*, arcul va fi umplut ca un semicerc deschis (OPEN), un semicerc închis (CHORD) sau un segment circular închis (PIE). Sursa poate fi schimbată funcția `ellipseMode()`. Diferența dintre aceste regimuri de desenare este arătată în figura 1.2.

Arcul întotdeauna este desenat în sensul acelor ceasornicului punctul de început (start) și săvârșit (stop) pot folosi constantele p5.js conform valorilor indicate în circumferință prezentată în figura 1.3. Dacă punctele start și stop cad în același loc, se va desena o elipsă (cerc) completă. Rețineți că axa Y crește în direcția în jos, astfel încât unghiurile sunt măsurate în sensul acelor de ceasornic din direcția X pozitivă.



**Figura 1.3. Relația dintre constante, radiani și grade în p5.js**

### Sintaxă:

`arc (x, y, w, h, start, stop, [mode], [detail])`

parametrii:

x (număr întreg): coordonata x a arcului;

y (număr întreg): coordonata y a arcului;

w (număr întreg): lățimea arcului;

h (număr întreg): înălțimea arcului;

start (număr întreg): unghiul de început a arcului;

stop (număr întreg): unghiul de sfârșit a arcului;

mode constantă: parametru care determină modul în care este desenat

arcul. COORD, PIE sau OPEN (optional);

detail (număr întreg): parametru optional numai pentru modul WebGL;



**Figura 1.2. Exemple de desenare a arcurilor**

**Funcția `ellipse()`:** desenează o elipsă (ovală) pe ecran. O elipsă cu lățime și înălțime egale este un cerc. În mod implicit, primii doi parametri specifică coordonatele centrului figurei, iar al treilea și al patrulea parametri specifică lățimea și înălțimea. Dacă nu este specificată înălțimea, valoarea lățimii este utilizată atât pentru lățime, cât și pentru înălțime. Dacă se specifică o înălțime sau o lățime negativă, se ia valoarea absolută. Sursa poate fi schimbată folosind funcția `ellipseMode()`.

#### Sintaxa:

```
ellipse(x, y, w, [h])
ellipse (x, y, w, h, detail)
```

parametrii:

x (număr întreg): coordonata x a centrului figurei;

y (număr întreg): coordonata y a centrului figurei;

w (număr întreg): lățimea elipsei;

h (număr întreg): înălțimea elipsei;

detail (număr întreg): parametru optional numai pentru modul WebGL;

**Funcția `circle()`:** desenează un cerc pe ecran. Această funcție este un caz particular al funcției `ellipse()` unde lățimea și înălțimea elipsei sunt aceleași. Înălțimea și lățimea elipsei corespund diametrului cercului. În mod implicit,

primii doi parametri stabilesc coordonatele centrului cercului, al treilea - diametrul cercului

**Sintaxă:**

`circle(x, y, d)`

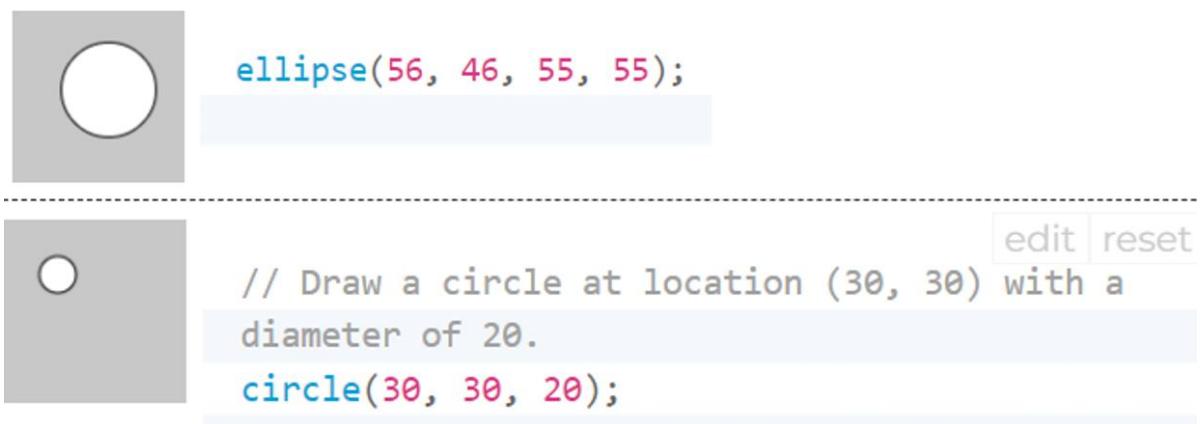
parametrii:

x (număr întreg): coordonata x a centrului figurei;

y (număr întreg): coordonata y a centrului figurei;

d (număr întreg): diametrul cercului;

În figura 1.3 este arătat un exemplu de utilizare a funcțiilor ellipse și circle.



**Figura 1.3. Exemple de program și rezultatul acestuia pentru funcțiile circle și ellipse**

**Funcția line():** trasează o linie dreaptă între două puncte pe ecran. Pentru a desena o linie de diferite grisimi, poate fi utilizat atributul stroke(). Linia nu poate fi umplută, deci atributul fill() nu va afecta aceasta. Liniile 2D sunt desenate implicit cu o lățime de un pixel, dacă dorim să specificăm grosimea liniei folosim funcția strokeWeight().

**Sintaxă:**

`line(x1, y1, x2, y2)`

`line(x1, y1, z1, x2, y2, z2)`

parametrii:

x1: x- coordonata primului punct;

y1: y- coordonata primului punct;

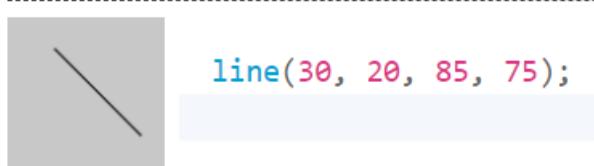
x2: x- coordonata punctului doi;

y2: y- coordonata punctului doi;

z1: coordonata z primului punct;

z2: coordonata z punctului doi;

Exemplu:



**Funcția point():** desenează un punct, o coordonată în spațiu, cu dimensiunea de un pixel. Primul parametru este valoarea orizontală a punctului, al doilea este valoarea verticală a punctului. Culoarea punctului poate fi schimbată folosind funcția stroke(). Mărimea punctului se modifică folosind funcția strokeWeight().

**Sintaxă:**

```
point(x, y, [z])  
point(coordinate_vector)
```

parametrii:

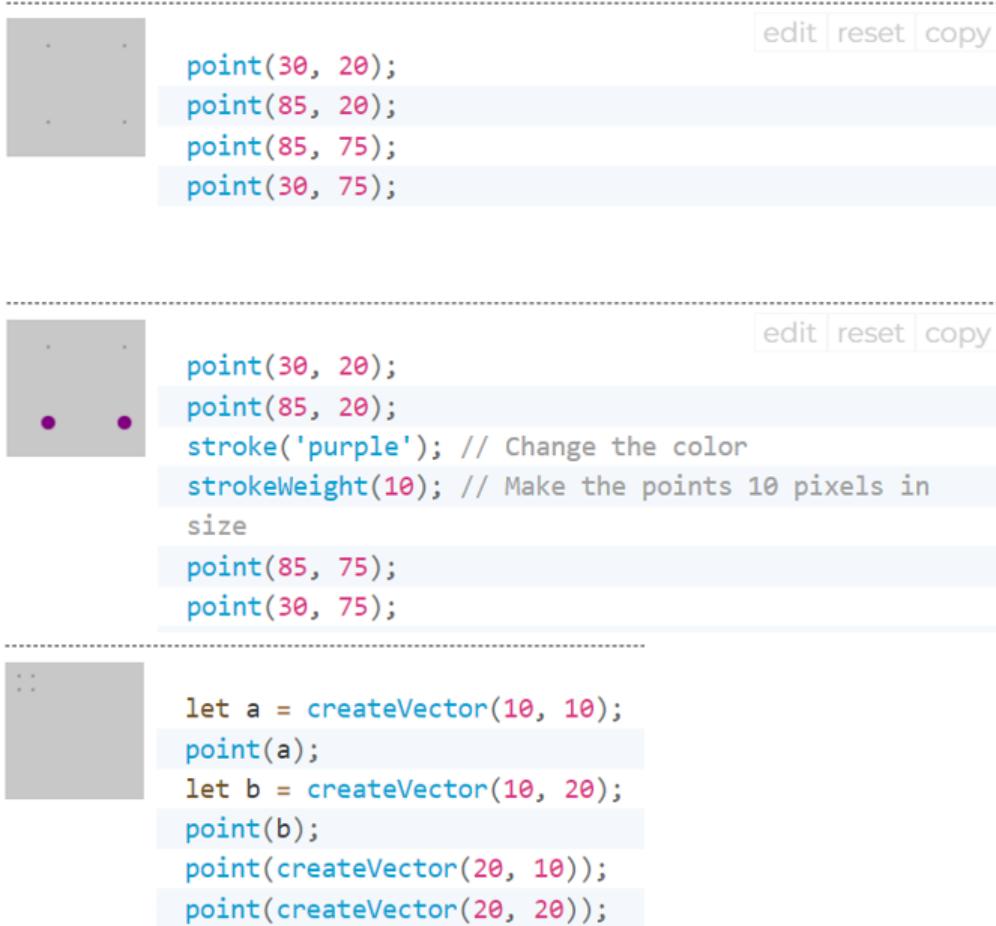
x: coordonata x;

y: coordonata y;

z: coordonata z (în regimul WebGL) (optional);

coordinate\_vector: un vector de coordonate;

Exemple de program și rezultatul acestuia pentru funcția point sunt reprezentate în figura 1.4



```
point(30, 20);  
point(85, 20);  
point(85, 75);  
point(30, 75);
```

```
point(30, 20);  
point(85, 20);  
stroke('purple'); // Change the color  
strokeWeight(10); // Make the points 10 pixels in size  
point(85, 75);  
point(30, 75);
```

```
let a = createVector(10, 10);  
point(a);  
let b = createVector(10, 20);  
point(b);  
point(createVector(20, 10));  
point(createVector(20, 20));
```

**Figura 1.4. Exemple de program pentru funcția point**

**Funcția quad():** desenează un patrulater (un poligon cu patru laturi), unghiurile dintre laturi nu sunt limitate la nouăzeci de grade. Prima pereche de parametri (x1, y1) specifică primul vârf, iar perechile ulterioare trebuie să se miște în sensul acelor ceasornicului sau în sens invers acestora în jurul unei

formei predefinite. Argumentele z funcționează numai când quad() este utilizat în modul WEBGL.

**Sintaxă:**

`quad(x1, y1, x2, y2, x3, y3, x4, y4)`  
`quad(x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4)`

parametrii:

x1: x- coordonata primului punct;  
y1: y- coordonata primului punct;  
x2: x- coordonata punctului doi;  
y2: y- coordonata punctului doi;  
x3: x- coordonata punctului trei;  
y3: y- coordonata punctului trei;  
x4: x- coordonata punctului patru;  
y4: y- coordonata punctului patru;  
z1: coordonata z primului punct;  
z2: coordonata z punctului doi;  
z3: coordonata z punctului trei;  
z4: coordonata z punctului patru;

**Funcția rect ()**: desenează un dreptunghi pe ecran, o figură cu patru laturi cu fiecare colț de nouăzeci de grade. În mod implicit, primii doi parametri specifică poziția colțului din stânga sus, al treilea parametru specifică lățimea, iar al patrulea parametru specifică înălțimea. Cu toate acestea, modul în care acești parametri sunt interpretați poate fi modificat folosind funcția rectMode().

Al cincilea, al șaselea, al șaptelea și al optulea parametri, dacă sunt specificați, definesc raza pentru colțurile din stânga sus, din dreapta sus, din dreapta jos și, respectiv, din stânga jos. Parametrul razei colțului este setat la valoarea razei specificate anterior în lista de parametri.

**Sintaxă:**

`rect(x, y, w, h, [tl], [tr], [br], [bl])`  
`rect(x, y, w, h, [detailX], [detailY])`

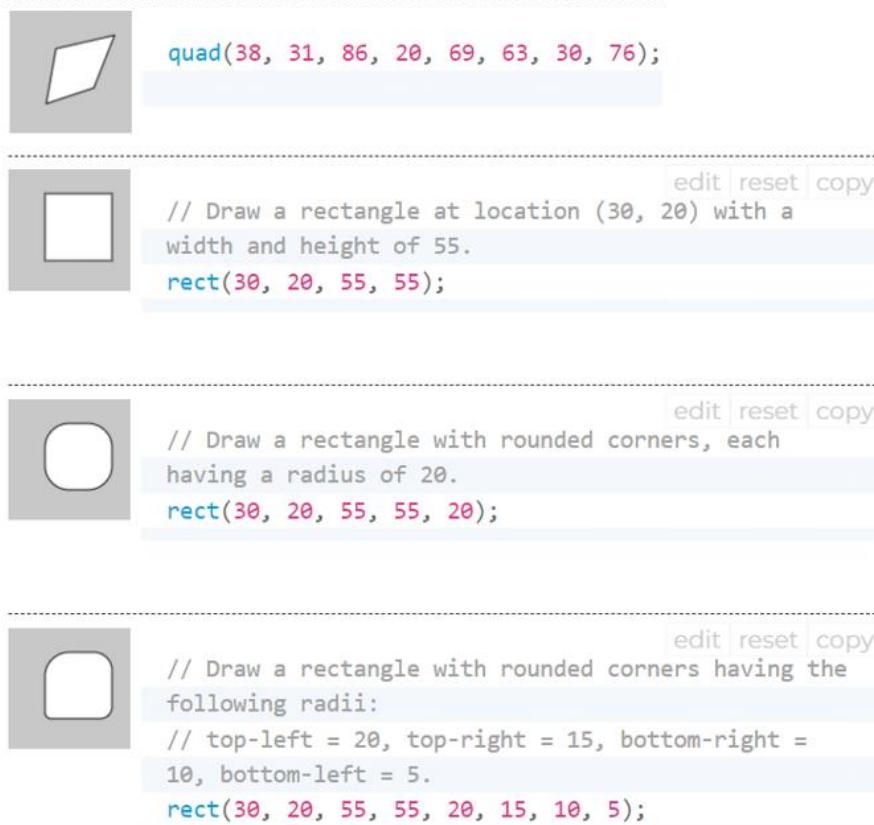
parametrii:

x: coordonata x a dreptunghiului;  
y: coordonata y a dreptunghiului;  
w : lățimea dreptunghiului;  
h: înălțimea dreptunghiului;  
tl: raza colțului stânga-sus (optional);  
tr: raza colțului dreapta-sus (optional);  
br: raza colțului dreapta-jos (optional);  
bl: raza colțului stânga-jos (optional);

detailX (număr întreg): numărul de segmente pe axa X (pentru regimul WebGL) (optional);

detailY (număr întreg): numărul de segmente pe axa Y (pentru regimul WebGL) (optional);

Exemple de program și rezultatele acestora sunt arătate în figura 1.5



```
quad(38, 31, 86, 20, 69, 63, 30, 76);  
  
// Draw a rectangle at location (30, 20) with a  
// width and height of 55.  
rect(30, 20, 55, 55);  
  
// Draw a rectangle with rounded corners, each  
// having a radius of 20.  
rect(30, 20, 55, 55, 20);  
  
// Draw a rectangle with rounded corners having the  
// following radii:  
// top-left = 20, top-right = 15, bottom-right =  
// 10, bottom-left = 5.  
rect(30, 20, 55, 55, 20, 15, 10, 5);
```

**Figura 1.5. Exemple de programuși pentru funcțiile rect și quad**

**Funcția square():** desenează un pătrat pe ecran, cu fiecare colț de nouăzeci de grade. Această funcție este un caz special al funcției rect() în care lățimea și înălțimea sunt aceleași, iar parametrul s pentru dimensiunea laturei. În mod implicit, primii doi parametri coordonata colțului din stânga sus, al treilea - dimensiunea laturii pătratului. Cu toate acestea, modul în care acești parametri sunt interpretați poate fi modificat folosind funcția rectMode().

Al patrulea, al cincilea, al șaselea și al șaptelea parametrii, dacă sunt specificați, definesc raza pentru colțurile din stânga sus, din dreapta sus, din dreapta jos și, respectiv, din stânga jos. Parametrul razei colțului este setat la valoarea razei specificate anterior în lista de parametri.

#### Sintaxa:

**square(x, y, s, [tl], [tr], [br], [bl])**

parametrii:

x: coordonata x a pătratului;

y: coordonata y a pătratului;

s: mărimea laturei pătratului;

tl: raza colțului stânga-sus (optional);

tr: raza colțului dreapta-sus (optional);

br: raza colțului dreapta-jos (optional);

bl: raza colțului stânga-jos (optional);



```

edit reset copy
// Draw a square at location (30, 20) with a side
size of 55.
square(30, 20, 55);

```



```

edit reset copy
// Draw a square with rounded corners, each having
a radius of 20.
square(30, 20, 55, 20);

```



```

edit reset copy
// Draw a square with rounded corners having the
following radii:
// top-left = 20, top-right = 15, bottom-right =
10, bottom-left = 5.
square(30, 20, 55, 20, 15, 10, 5);

```

**Figura 1.6. Exemplu de utilizare a funcției square**

**Funcția triangle():** desenează un triunghi. Argumentii funcției specifică coordonatele primul punct, coordonatele punctului doi, iar ultimele două argumente indică coordonatele punctului treilea al.

**Sintaxa:**

**triangle(x1, y1, x2, y2, x3, y3)**

parametrii:

- x1: x- coordonata primului punct;
- y1: y- coordonata primului punct;
- x2: x- coordonata punctului doi;
- y2: y- coordonata punctului doi;
- x3: x- coordonata punctului trei;
- y3: y- coordonata punctului trei;



```

triangle(30, 75, 58, 20, 86, 75);

```

**Figura 1.7 Exemplu de utilizare a funcției triangle**

## 1.2 Atribute grafice simple

Prticularitățile figurilor geometrice, cum at fi culoarea figurii, grosimea și culoarea liniei tipul hașirării, modul de conexiune a liniilor reprezintă attributele grafice simple. În biblioteca p5.js sunt o listă de funcții care permit setarea sau modificarea acestor attribute.

Principalele attributele grafice sunt descrise în continuare:

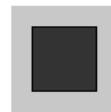
**Funcția fill():** Setează culoarea folosită pentru umplerea figurilor, toate figurile desenate după această funcție umplete (colorate) cu culoarea indicate ca parametru funcției. Această culoare este specificată în termeni de culoare RGB sau HSB, în funcție de colorMode() curent. (Spațiul de culoare implicit este RGB, fiecare valoare variază de la 0 la 255).

**Sintaxa:**

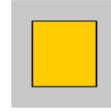
```
fill(v1, v2, v3, [alpha])
fill(value)
fill(gray, [alpha])
fill(values)
fill(color)
```

parametrii:

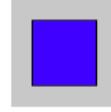
v1 (număr întreg):	roșu sau o valoare nuanței în raport cu gama de culori curentă;
v2 (număr întreg):	verde sau valoare saturăției în raport cu gama de culori curentă;
v3 (număr întreg):	albastru sau valoarea luminozității în raport cu gama de culori curentă;
alpha (număr întreg):	(optional);
value (un string):	string care indică culoarea;
gray (număr întreg):	valoarea culorii sure;
values (numere []):	un tabel care conține componentele culorilor:rușii, verde, albastră și alpha;



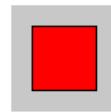
```
// Grayscale integer value
fill(51);
rect(20, 20, 60, 60);
```



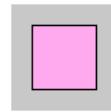
```
// R, G & B integer values
fill(255, 204, 0);
rect(20, 20, 60, 60);
```



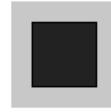
```
// H, S & B integer values
colorMode(HSB);
fill(255, 204, 100);
rect(20, 20, 60, 60);
```



```
// Named SVG/CSS color string
fill('red');
rect(20, 20, 60, 60);
```



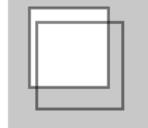
```
// three-digit hexadecimal RGB notation
fill('#fae');
rect(20, 20, 60, 60);
```



```
// six-digit hexadecimal RGB notation
fill('#222222');
rect(20, 20, 60, 60);
```

**Figura 1.8 a) Exemple de utilizare a funcției fill**

Dacă dorim ca figura să fie transparentă atunci utilizăm funcția **nofill**.



```
rect(15, 10, 55, 55);
noFill();
rect(20, 20, 60, 60);
```

**Figura 1.8 b) Exemple de utilizare a funcției nofill**

**Funcția stroke ():** Specifică culoarea folosită pentru a desena liniile și marginile figurilor. Această culoare este specificată în termeni de culoare RGB sau HSB, în funcție de colorMode() curent (spațiul de culoare implicit este RGB, fiecare valoare variază de la 0 la 255). Intervalul alfa implicit este, de asemenea, de la 0 la 255.

**Sintaxa:**

**stroke(v1, v2, v3, [alpha])**  
**stroke(value)**  
**stroke(gray, [alpha])**

`stroke(values)`

`stroke(color)`

parametrii:

v1 (număr întreg): roșu sau o valoare nuanței în raport cu gama de culori curentă;

v2 (număr întreg): verde sau valoare saturației în raport cu gama de culori curentă;

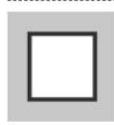
v3 (număr întreg): albastru sau valoarea luminozității în raport cu gama de culori curentă;

`alpha` (număr întreg): (optional)

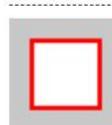
`value` (un string): string care indică culoarea;

`gray` (număr întreg): valoarea culorii sure;

`values` (numere []): un tabel care conține componentele culorilor:rușii, verde, albastră și alpha;



```
// Grayscale integer value  
strokeWeight(4);  
stroke(51);  
rect(20, 20, 60, 60);
```



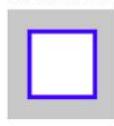
```
// Named SVG/CSS color string  
stroke('red');  
strokeWeight(4);  
rect(20, 20, 60, 60);
```



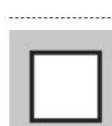
```
// R, G & B integer values  
stroke(255, 204, 0);  
strokeWeight(4);  
rect(20, 20, 60, 60);
```



```
// three-digit hexadecimal RGB notation  
stroke('#fae');  
strokeWeight(4);  
rect(20, 20, 60, 60);
```



```
// H, S & B integer values  
colorMode(HSB);  
strokeWeight(4);  
stroke(255, 204, 100);  
rect(20, 20, 60, 60);
```



```
// six-digit hexadecimal RGB notation  
stroke('#222222');  
strokeWeight(4);  
rect(20, 20, 60, 60);
```

**Figura 1.9 a). Exemple de utilizare a funcției `stroke`**

Dacă dorim ca figura să fie desenată fără margine atunci utilizăm funcția `nostroke`.



```
noStroke();  
rect(20, 20, 60, 60);
```

**Figura 1.9 b). Exemplu de utilizare a funcției `nostroke`**

În afara culorii funcția `stroke` poate specifica următoarele:

`strokeCap()`

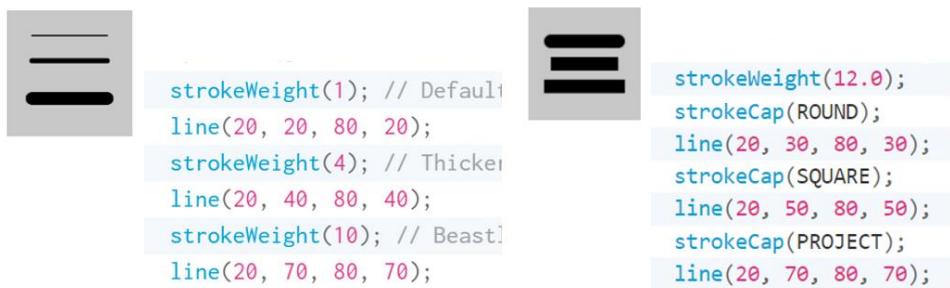
`strokeJoin()`

## strokeWeight()

**Funcția strokeWeight(number):** Setează lățimea liniei. Toate valorile sunt specificate în pixeli.

**Funcția strokeCap(cap):** Setează stilul de redare a sfârșitului liniei. Aceste capete sunt fie rotunjite, pătrate sau extinse, fiecare dintre acestea fiind specificat cu parametrii corespunzători: ROUND, SQUARE și PROJECT. Implicit este ROUND.

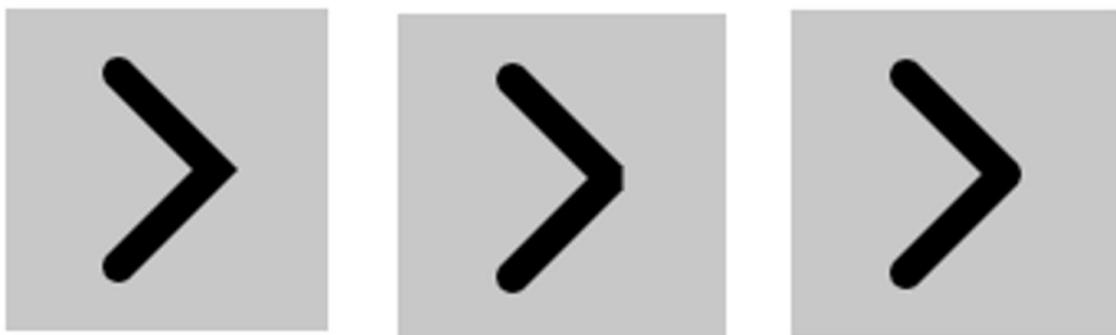
Exemple de utilizare a acestor argumente sunt prezentate în figura 1. 10.



**Figura 1.10. Exemple de utilizare a funcției stroke**

**Funcția strokeJoin(join):** Setează stilul conectării (unirii) segmentelor de linie. Aceste pot fi specificate cu parametrii corespunzători MITRE, BEVEL și ROUND. Implicit este setat ca MITRE.

Diferența dintre aceste moduri este prezentată în figura 1.11.



**Figura 1.11 Parametrii funcției strokeJoin**

În cazul în care este nevoie de adăugat text pot fi utilizată funcția text care poate avea diferenți parametrii.

**Funcția text():** Desenează text pe ecran, afișează informațiile specificate în primul parametru pe ecran în poziția specificată de parametrii suplimentari. Se va folosi un font implicit, cu excepția cazului în care un font este setat cu funcția **textFont()** și o dimensiune implicită va fi folosită dacă nu este setat un font cu **textSize()**. Culoarea textului poate fi schimbată cu funcția **fill()**. Schimbarea conturului textului se efectuează cu funcțiile **stroke()** și **strokeWeight()**.

Textul este afișat în relație cu funcția **textAlign()**, care oferă opțiunea de a desena la stânga, la dreapta și în centrul coordonatelor.

### Sintaxa:

**text(str, x, y, [x2], [y2])**

Parametrii:

str : se indică sirul pentru afișare pe ecran;

x: coordonata x a punctului de început a textului;

y: coordonata y a punctului de început a textului;

x2 și y2: definesc o zonă dreptunghiulară în care să se afișeze și pot fi utilizați numai cu date de tip sir de caractere. Când acești parametri sunt specificați, ei sunt interpretați pe baza setării curente **rectMode()**. Textul care nu se încadrează complet în dreptunghiul specificat nu va fi desenat pe ecran. Dacă x2 și y2 nu sunt specificate, alinierea liniei de bază este implicită, ceea ce înseamnă că textul va fi desenat în sus de la x și y.

Principalele atributele textului sunt aduse în figura 1.12



**Figura 1.12 Exemple de utilizare a funcțiilor de lucru cu textul**

## 1.3 Formatele grafice

Orice imagine grafică este salvată în fișier. Modul în care datele grafice sunt stocate într-un fișier determină formatul grafic al fișierului.

**Format**- structura fișierului, care determină modul în care sunt stocate și afișate pe ecran sau la imprimare datele. Formatul fișierului este de obicei indicat în numele său, ca o parte separată printr-un punct (de obicei, această parte se numește extensia numelui fișierului).

Compresia este utilizată pentru fișierele grafice, deoarece au un volum destul de mare de informație pe care o conțin, fiecare format are propriul algoritm prin care sunt comprimate datele conținute în fișier.

Formatele grafice se clasifică după:

- - tipul datelor stocate (raster, vector și forme mixte),
- - în funcție de cantitatea admisă de date

- - parametrii imaginii
- - modul de stocare a paletei de culori
- - metoda de compresie a datelor
- - prin metode de organizare a fișierelor (text, binar)

Din varietatea de formate, nu există niciunul ideal care să satisfacă toate cerințele posibile. Alegerea unuia sau a altui format pentru salvarea unei imagini depinde de obiectivele și scopuri de lucru cu imaginea. Dacă aveți nevoie de o acuratețe fotografică a culorilor, atunci este preferat unul dintre formatele raster. Pentru sigle, diagrame, elemente de design sunt recomandabile formate de stocare vectoriale. Formatul fișierului afectează cantitatea de memorie pe care o ocupă fișierul. Editorii grafici permit utilizatorului să aleagă independent formatul pentru salvarea imaginii. Există formate de fișiere grafice universale care acceptă atât imagini vectoriale, cât și imagini bitmap în același timp.

În tabelul 1.1 este arătată o scurtă descriere a formatelor fișierelor grafice utilizate.

**Tabelul 1.1. Caracteristicile formatelor grafice**

Format	Mod imagine	Tipul informațiilor grafice	Cerere
BMP	Numai culorile indexate	Modele de tipul aplicațiilor care conțin zone întinse de culoare solidă.	Formatul este acceptat de toate aplicațiile. Nu este utilizat în publicare din cauza volumului mare de fișiere.
Tiff	Tot	Imagini de tip diagramă	Un format universal pentru stocarea imaginilor scanate cu canale color. Include scheme de compresie pentru a reduce dimensiunea fișierului. Un avantaj important al formatului este portabilitatea sa pe diferite platforme. În mod tradițional, TIFF poate fi considerat formatul preferat pentru realizarea de machete axate pe tipărire tipografică și alte metode de reproducere.
PSD	Suportă toate tipurile de imagini	Orice imagini	Intern pentru program Adobe Photoshop. Singurul format în care sunt salvate toate informațiile despre un document, inclusiv straturile și canalele. Cu toate acestea, este mai bine să salvați imaginea terminată în alte formate grafice din două motive. La început, Fișier PSD cu

			dimensiuni mult mai mari. În al doilea rând, acest format nu este importat de aspectul și programele de grafică obiect.
Jpeg	Imagini color complet doar la modelele RGB și CMYK	Fotografii depline sau mostre de grafică artistică, inclusiv revărsări subtile de culori.	Concepțut pentru a salva fișiere punct cu compresie. Comprimarea utilizând această metodă reduce dimensiunea fișierului de la zeci de la sută la o sută de ori (interval practic - de la 5 la 15 ori), dar compresia în acest format are pierderi (în limite acceptabile). Un algoritm de compresie foarte eficient a condus la cea mai largă distribuție a JPEG pe World Wide Web. Nu este recomandată utilizarea acestui format în industria tipografică.
GIF	Numai imagini indexate	Desene de tip diagramă - imaginile au suprafete mari de culoare uniformă cu limite bine definite; imagini animate	Proiectat special pentru transmiterea imaginilor în rețelele globale. Are cea mai eficientă metodă de compresie, care este necesară pentru a reduce timpul de transmisie a imaginilor. O nouă versiune permite stocarea mai multor imagini într-un singur fișier. Cea mai obișnuită utilizare a acestei caracteristici este pe web. Browserul web afișează imagini situate în Fișier GIF, secvențial.
Imagine PNG	Suportă imagini color RGB și imagini indexate.	Imagini color cu tranziții fluide de la zone opace la zone transparente	Numele formatului, Portable Network Graphics, vorbește despre scopul său - de a transfera imagini pe rețele. Este posibil să utilizați un singur canal suplimentar pentru stocarea măștii de transparentă. Are un algoritm de compresie eficient, fără pierderi de informații. Formatul este utilizat pe web.
EPS	Tot	Grafică vectorială, fonturi, imagini rasterizate	Este utilizat în industria tipografică. Este posibil să stocați informații

			despre rasterizare, contururi și curbe de calibrare.
--	--	--	--

### **Surse bibliografice:**

- <https://p5js.org/reference/>
- <https://github.com/processing/p5.js/wiki/p5.js-overview>
- "Make: Getting started with p5.js" Lauren McCarthy, Casey Reas, Ben Fry;
- "Learn JavaScript with p5.js" Engin Arslan

## **Lucrarea de laborator 1**

### **Tema: Studierea primitivelor grafice simple 2D**

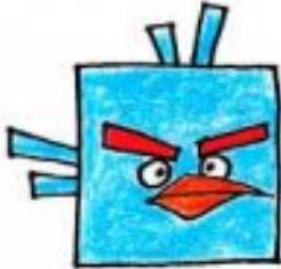
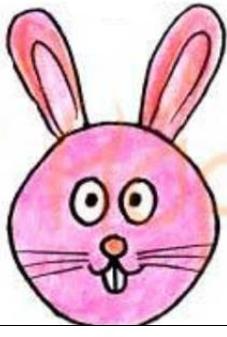
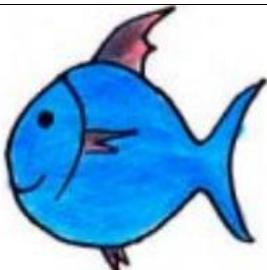
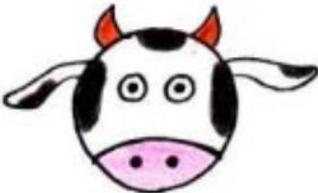
**Scopul lucrării:** Obținerea cunoștințelor practice în sinteza scenelor grafice 2D statice, utilizând primitivele grafice simple a bibliotecii p5.js.

#### **Sarcina lucrării:**

1. Elaborați un program pentru sinteza unei scene 2D statice utilizând cel puțin 6 primitive grafice de diferite cum ar fi - *arc()*, *ellipse()*, *circle()*, *line()*, *point()*, *quad()*, *rect()*, *square()*, *triangle()*, primitivele trebuie să fie cu diferite atrubute, lucrarea trebuie semnată (numele prenumele grupa) în colțul dreapta jos a ecranului.

2. Elaborați un program care crează personajul conform variantei indicate de profesor. Variantele sunt indicate în tabelul 1.2. Pentru crearea acestei imagini pe pași puteți consulta pagina <https://jarrastu.ru/raskraski/uroki/1079-poshagovoe-risovanie-zhivotnyh-iz-geometricheskikh-figur.html>

**Tabelul 1.2 Variantele pentru realizarea lucrarii de laborator**

Varianta	Personajul	Varianta	Personajul
1		11	
2		12	
3		13	
4		14	
5		15	
6		16	

7		17	
8		18	
9		19	
10		20	

### Exemplu de program realizat în p5.js:

```

var Width;
var Height;
var CurrentY;

function setup() {
  Width = 400;
  Height = 734;
  createCanvas(Width, Height);
}

function draw() {
  background(220);
  CurrentY = Height - 20;

  //Realizam baza (1 parte)
  stroke(6, 21, 131);
  for (let i = 0; i < 20; i++) {
    line(0 + 20, CurrentY, Width - 20, CurrentY);
    CurrentY--;
  }

  // Realizam baza (2 parte)
  stroke(15, 23, 71);
  for (let i = 0; i < 20; i++) {

```

```

line(0 + 20 + i, CurrentY, Width - 20 - i, CurrentY);
CurrentY--;
}

//Основание двери
CurrentY += 10;
stroke(6, 21, 121);
for (let i = 0; i < 550; i++) {
    line(0 + 40, CurrentY, Width - 40, CurrentY);
    CurrentY--;
}

//Колонки
fill(6, 21, 121);
stroke(15, 21, 71);
rect(40, CurrentY, 40, Height - 190);
rect(80, CurrentY, 3, Height - 190);
rect(83, CurrentY, 6, Height - 190);

rect(Width - 40, CurrentY, -40, Height - 190);
rect(Width - 80, CurrentY, -3, Height - 190);
rect(Width - 83, CurrentY, -6, Height - 190);

//Дверные ямы
for (let i = 0; i < 4; i++) {
    for (let j = 0; j < 2; j++) {
        stroke(129, 153, 193);
        line(110 + j * 100, Height - 80 - i * 130, 110 + j * 100, Height - 180 - i * 130);
        line(110 + j * 100, Height - 80 - i * 130, 190 + j * 100, Height - 80 - i * 130);
        stroke(10, 14, 45);
        line(110 + j * 100, Height - 180 - i * 130, 190 + j * 100, Height - 180 - i * 130);
        line(190 + j * 100, Height - 180 - i * 130, 190 + j * 100, Height - 80 - i * 130);
    }
}

//Средняя колонка
stroke(10, 14, 45);
rect(200, CurrentY, -3, Height - 190);
stroke(16, 31, 138);
rect(203, CurrentY, -3, Height - 190);
stroke(49, 65, 157);
rect(205, CurrentY, -1, Height - 190);

//Дверь
fill(240, 240, 240);
ellipse(210, 350, 8, 30);
fill(147, 127, 68);
circle(210, 410, 10);

stroke(10, 14, 45);
line(110, Height - 80 - 2 * 130, 110, Height - 180 - 2 * 130);
line(110, Height - 80 - 2 * 130, 190, Height - 80 - 2 * 130);
line(110, Height - 180 - 2 * 130, 190, Height - 180 - 2 * 130);
line(190, Height - 180 - 2 * 130, 190, Height - 80 - 2 * 130);
fill(240, 240, 240);
stroke(240, 240, 240);
rect(120, Height - 430, 60, 80);
fill(0, 0, 0);
noStroke();
textSize(5);
text('POLICE TELEPHONE', 125, Height - 420);
textSize(10);

```

```

text('FREE', 135, Height - 405);
textSize(5);
text('FOR USE OR', 132, Height - 395);
textSize(10);
text('PUBLIC', 130, Height - 380);
textSize(5);
text('ADVICE & ASSIS', 128, Height - 370);
textSize(7);
text('PULL TO OPEN', 125, Height - 355);

//Окна
for(let j = 0; j < 2; j++) {
  stroke(129, 153, 193);
  fill(240, 240, 240);
  rect(113 + j * 100, Height-565, 75,93);

  stroke(18, 34, 129);
  line(138 + j * 100, Height-565, 138 + j * 100,Height-473);
  line(163 + j * 100, Height-565, 163 + j * 100,Height-473);
  line(113 + j * 100, Height-519, 188 + j * 100,Height-519);
}

//Верхушка
CurrentY-=40
fill(6, 21, 121);
stroke(15, 21, 71);
rect(35, CurrentY, 330,50);
stroke(3, 11, 101);
strokeWeight(10);
fill(22, 27, 46);
rect(65, CurrentY, 270,50);
strokeWeight(1);

fill(255,255,255);
noStroke();
textSize(26);
text('POLICE', 90, CurrentY+35);
text('BOX',260, CurrentY+35);
textSize(12);
text('PUBLIC', 200, CurrentY+25);
text('CALL', 209, CurrentY+39);

CurrentY-=30
fill(6, 21, 121);
stroke(15, 21, 71);
rect(65, CurrentY, 270,30);

CurrentY-=20
rect(85, CurrentY, 230,20); }

```

Rezultatul realizării programului:



### ***Întrebări de control:***

1. Numiți primitive grafice simple.
2. Cum poate fi realizată modificarea atributelor de afișare ale primitivelor grafice?
3. Cum poate fi scris textul în mod grafic?
4. Numiți formate standard pentru imagini.

### **Transformări grafice 2D**

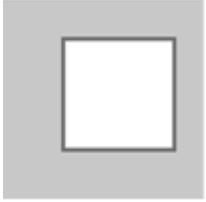
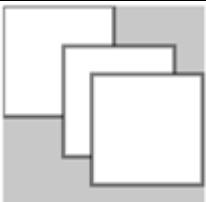
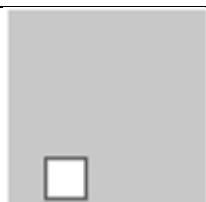
**Transformări grafice 2D în biblioteca grafică p5jc:**

- (translate)
- (scale)
- (rotate)

### **Translarea**

Funcția **translate()** permite mutarea obiectelor în orice locație din fereastră. În **Tabelul 2.1** este indicate sintaxa translației.

**Tabelul 2.1**

	<code>translate(30, 20); rect(0, 0, 55, 55);</code>
	<code>rect(0, 0, 55, 55); // Draw rect at original 0,0 translate(30, 20); rect(0, 0, 55, 55); // Draw rect at new 0,0 translate(14, 14); rect(0, 0, 55, 55); // Draw rect at new 0,0</code>
	<code>function draw() {     background(200);     rectMode(CENTER);     translate(width / 2, height / 2);     translate(p5.Vector.fromAngle(millis() / 1000,         40));     rect(0, 0, 20, 20); }</code>

În exemplul de mai jos veți vedea metoda de executare a translației în cod:

```
let x = 0;
let y = 0;
let dim = 80.0;

function setup() {
  createCanvas(720, 400);
  noStroke();
}

function draw() {
  background(102);

  // Animează creșterea valorii x
  x = x + 0.8;

  // Dacă forma ieșe din grilă, resetați poziția
  if (x > width + dim) {
    x = -dim;
  }

  // Chiar dacă comanda noastră „rect” desenează forma cu centrul său
  // la origine, translația o mută în noua poziție x și y
```

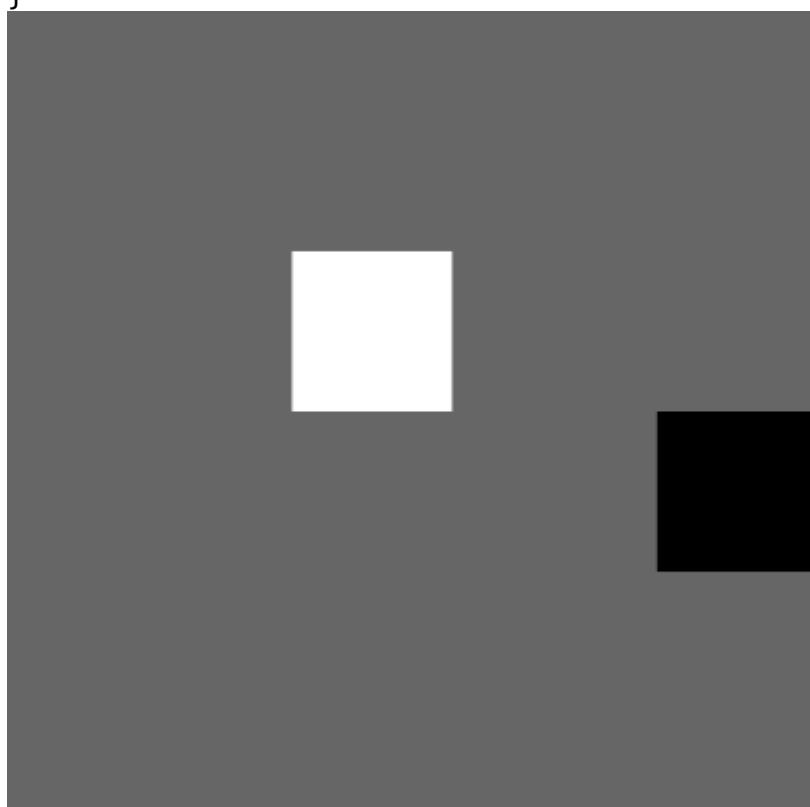
```

translate(x, height / 2 - dim / 2);
fill(255);
rect(-dim / 2, -dim / 2, dim, dim);

// Transformările se acumulează. Observați cum acest „rect”
// se mișcă de două ori mai repede decât celălalt, dar are
// același parametru pentru valoarea axei x

translate(x, dim);
fill(0);
rect(-dim / 2, -dim / 2, dim, dim);
}

```

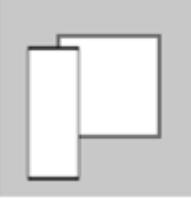


**Figura 2.1 Translarea**

## Scalarea

Funcția **scale()** transformare care redimensionează un element în planul 2D. Deoarece cantitatea de scalare este definită de un vector, acesta poate redimensiona dimensiunile orizontale și verticale la scări diferite. În **Tabelul 2.2** este indicate sintaxa scalării.

**Tabelul 2.2**

	<pre>rect(30, 20, 50, 50); scale(0.5); rect(30, 20, 50, 50);</pre>
	<pre>rect(30, 20, 50, 50); scale(0.5, 1.3); rect(30, 20, 50, 50);</pre>

Parametrii pentru funcția **scale()** sunt valori specificate ca procente zecimale. De exemplu, metoda scala de apel (2.0) va crește dimensiunea formei cu 200 la sută. Obiectele se întind întotdeauna de la origine. Acest exemplu arată cum se acumulează transformările și, de asemenea, cum interacționează scala și interschimbarea în funcție de ordinea lor. În exemplul de mai jos veți o metodă de executare a translației în cod:

```
let a = 0.0;
let s = 0.0;

function setup() {
  createCanvas(720, 400);
  noStroke();

  // Desenați toate dreptunghiurile din centrul lor
  rectMode(CENTER);
}

function draw() {
  background(102);

  // Creșteți „a” și apoi animați „s” cu
  // o mișcare ciclică prin găsirea cosinusului „a”

  a = a + 0.04;
  s = cos(a) * 2;

  // Translați dreptunghiul de la origine la mijlocul
  // grlei, apoi scalăți-l cu „s”

  translate(width / 2, height / 2);
  scale(s);
  fill(51);
  rect(0, 0, 50, 50);

  // Translarea și scala se acumulează, această translare
```

```

// deplasează al doilea dreptunghi mai la dreapta decât primul
// și scara se dublează. Rețineți că cosinusul este
// făcând „s” atât negativ, cât și pozitiv, astfel se ciclează
// de la stanga la dreapta.

translate(75, 0);
fill(255);
scale(s);
rect(0, 0, 50, 50);
}

```

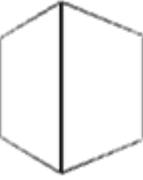


**Figura 2.2 Translarea  
Rotire**

Funcția de rotire definește o transformare care rotește un element în jurul unui punct fix pe planul 2D, fără a-l deformă. Rezultatul său este un tip de date **rotate()**

**Tabelul 2.3**

	<pre> translate(width / 2, height / 2); rotate(PI / 3.0); rect(-26, -26, 52, 52); </pre>
	<pre> function setup() {   createCanvas(100, 100, WEBGL); } function draw() {   background(255);   rotateX(millis() / 1000); } </pre>

	<pre>         box();     }  function setup() {     createCanvas(100, 100, WEBGL); }  function draw() {     background(255);     rotateY(millis() / 1000);     box(); } </pre>
	<pre> function setup() {     createCanvas(100, 100, WEBGL); }  function draw() {     background(255);     rotateZ(millis() / 1000);     box(); } </pre>

Rotirea unui pătrat în jurul axei Z. Pentru a obține rezultatele pe care le așteptați, trimiteți parametrii unghiului funcției de rotație care sunt valori cuprinse între 0 și PI \* 2 (TWO\_PI care este de aproximativ 6,28). Dacă preferați să vă gândiți la unghiuri ca grade (0-360), puteți utiliza metoda radians () pentru a vă converti valorile. De exemplu: rotate (radians (90)) este identic cu declarația rotate (PI / 2). În acest exemplu, la fiecare secundă numerotată pară se adaugă o rotație la rotație. În secunde impare, rotația deplasează CW și CCW la viteza determinată de ultima valoare a jitterului.

```

let angle = 0.0;
let jitter = 0.0;

function setup() {
    createCanvas(720, 400);
    noStroke();
    fill(255);

    // Desenați dreptunghiul din centru care va fi și
    // rotația în jurul acelui centru

    rectMode(CENTER);
}

function draw() {
    background(51);

    // în secunde par (0, 2, 4, 6...) adăugați jitter la
    // rotație

    if (second() % 2 === 0) {

```

```

        jitter = random(-0.1, 0.1);
    }

    // măriți valoarea unghiului folosind
    // cea mai recentă valoare de jitter

    angle = angle + jitter;

    // utilizați cosinusul pentru a obține
    // o mișcare lină CW și CCW atunci când nu vibrați

    let c = cos(angle);

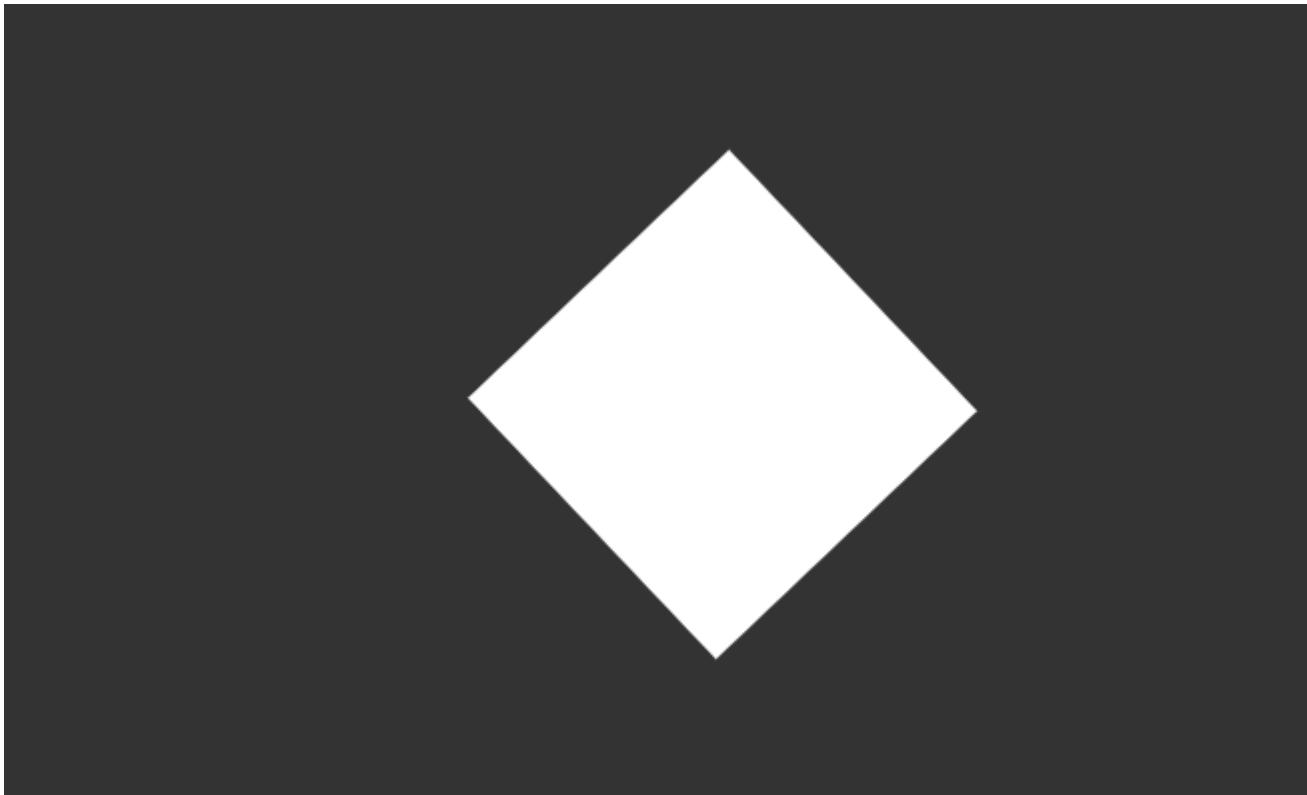
    // mutați forma în centrul grilei

    translate(width / 2, height / 2);

    // aplicați rotația finală

    rotate(c);
    rect(0, 0, 180, 180);
}

```



**Figura 2.3 Rotire**

## **Lucrare de laborator nr.2**

### **Tema:** Transformări grafice 2D

**Scopul:** Implementarea transformărilor grafice asupra unei scene 2D utilizînd setul de funcționalități a bibliotecii (JavaScript) p5.js.

**Sarcina:** Elaborați un program pentru efectuarea transformărilor grafice 2D utilizând **rotate()**, **scale()**, **translate()**.

Transformările vor fi aplicate asupra scenei 2D create la lucrare de laborator nr.1.

Exemplu de program pentru fiecare transformare le puteți găsi mai sus în Tabelul 2.1, Tabelul 2.2 și Tabelul 2.3.

**Tabelul 2.4 Variantele pentru realizarea lucrării de laborator**

<b>Variană</b>	<b>Unghiul de rotire</b>	<b>Coefficientul de scalare</b>	<b>Deplasarea X și Y</b>
<b>1</b>	320	0.5	10,10
<b>2</b>	70	1.2	20,10
<b>3</b>	30	2.1	15,10
<b>4</b>	25	0.8	100,100
<b>5</b>	100	1.3	90,85
<b>6</b>	110	1.4	15,15
<b>7</b>	20	0.6	80,80
<b>8</b>	49	1.1	45,40
<b>9</b>	95	0.8	20,25
<b>10</b>	64	1.9	70,70
<b>11</b>	38	0.2	35,35
<b>12</b>	128	2.3	40,45
<b>13</b>	300	1.47	100,110
<b>14</b>	256	0.3	90,95
<b>15</b>	49	1.3	20,10
<b>16</b>	60	1.29	10,20
<b>17</b>	83	1.7	40,20
<b>18</b>	39	0.4	85,100
<b>19</b>	26	1.23	30,35

<b>20</b>	74	1.97	20,30
<b>21</b>	91	2.3	105,125
<b>22</b>	90	1.32	110,130
<b>23</b>	95	1.90	100,90
<b>24</b>	44	1.5	120,115
<b>25</b>	79	0.9	20,40

## CAPITOLUL 3

### Utilizarea obiectelor 3D în scene statice

#### 3.1. Fișiere .OBJ

Formatul de fișier OBJ este unul dintre cele mai importante formate de fișiere în aplicațiile de imprimare 3D și grafică 3D. Este formatul preferat pentru imprimarea 3D multicolor și este utilizat pe scară largă ca format de schimb neutru pentru modelele 3D non-animate în aplicațiile grafice.

Un fișier OBJ este un format standard de imagine 3D care poate fi exportat și deschis de diverse programe de editare 3D. Acesta conține un obiect tridimensional, care include coordonatele 3D, hărțile de texturi, fețele poligonale și alte informații despre obiect.

Pe scurt, formatul de fișier OBJ stocă informații despre modelele 3D. Poate codifica geometria suprafeței unui model 3D și poate stoca, de asemenea, informații despre culoare și textură. Acest format nu stocă nicio informație despre scenă cum ar fi poziția luminii sau animației.

Un fișier OBJ este de obicei generat de un software CAD (Computer Aided Design) ca produs final al procesului de modelare 3D. Extensia de fișier corespunzătoare formatului de fișier OBJ este pur și simplu „.OBJ”.

Formatul de fișier OBJ este open source și neutru. Este adesea utilizat pentru partajarea modelelor 3D în aplicații grafice, deoarece se bucură de un bun suport de import și export din aproape toate software-urile CAD. Acesta este de asemenea utilizat ca format de fișier pentru imprimarea 3D multicolor, deoarece formatul standard de imprimare 3D STL nu include informații despre culoare și textură.

Formatul de fișier OBJ a fost creat inițial de Wavefront Technologies pentru aplicația sa Advanced Visualizer pentru a stoca obiecte geometrice compuse din linii, poligoane și curbe și suprafețe de formă liberă. Cea mai recentă versiune documentată este v3.0, înlocuind versiunea anterioară v2.11.

Cel mai dominant format din lumea tipăririi 3D este STL. Cu toate acestea, STL este un format de fișier vechi care, deși este foarte popular, nu a ținut cont de cerințele moderne. Exactitatea imprimării 3D se apropie rapid de rezoluția la nivel de micron și modelele multicolore devin din ce în ce mai populare. Formatul de fișier STL nu poate suporta destul de bine rezoluțiile mari, deoarece o rezoluție mai mare duce la creșterea dimensiunii fișierului. De asemenea, formatul STL nu este potrivit pentru imprimarea 3D multicolor, deoarece nu suportă informații despre culoare și textură. În schimb, formatul OBJ poate aproxima geometria suprafeței destul de precis, fără a avea un impact semnificativ asupra dimensiunii fișierului. Acest lucru este posibil folosind curbele Bezier și o metodă numită NURBS. În plus, formatul de fișier OBJ are suport nativ pentru mai multe culori și texturi din același model.

Astfel, utilizarea formatului OBJ are o serie de avantaje în comparație cu utilizarea formatului STL în cazul în care aveți nevoie de modele cu rezoluție înaltă, multi-color. Pe de altă parte, formatul de fișier OBJ nu este la fel de universal ca formatul STL. Aproape toate imprimantele 3D acceptă formatul STL. Nu se poate spune același lucru și despre formatul OBJ, chiar dacă se bucură de suport. Prin urmare, dacă este nevoie de a tipări un model 3D monocrom la o imprimantă standard, este preferabil formatul STL.

Ambele formate OBJ și STL au un spectru larg de utilizare, cu o bază mare de utilizatori fideli și se bucură de suportul unei mulțimi de aplicații terțe.

Concurenții principali în cazul formatelor de fișiere pentru imprimare 3D sunt VRML, AMF și 3MF, care însă nu se bucură de un așa support și compatibilitate și deci nu sunt alternative serioase ale formatelor de fișiere STL și OBJ.

Format fișier OBJ se utilizează în aplicații grafice 3D. Cele mai utilizate formate de fișiere în aplicațiile 3D Graphics sunt OBJ, FBX și COLLADA.

### Diferențe între formatele de fișiere OBJ, FBX, COLLADA

Cea mai importantă diferență între formatul de fișier OBJ și celelalte două (FBX și COLLADA) este suportul pentru informații despre scenă cum ar fi sursele de lumină și animații. Formatul de fișier OBJ nu conține informații despre scene și animații, în timp ce FBX și COLLADA o fac. Prin urmare, dacă se dorește întreținerea suportului pentru animații în cazul jocurilor sau filmurilor, atunci se recomandă utilizarea formatului FBX sau COLLADA. În cazul în care însă nu este nevoie de o scenă complexă sau de animații, poate fi argumentată utilizarea formatului de fișier OBJ.

### Avantajele formatului obj

În primul rând, formatul de fișier OBJ este un format simplu și deschis. Are suport larg pentru export și import printre software CAD. Aceasta înseamnă că dacă partajați modelul 3D ca fișier OBJ, atunci alte programe CAD îl vor interpreta corect și consecvent. Nu se poate spune același lucru despre formatele FBX sau COLLADA. Formatul COLLADA este, de asemenea, open source, dar este destul de complicat. Diferite software CAD îl interpretează diferit și acest lucru poate duce la erori.

Formatul FBX este un format închis și proprietar și oferă un SDK pentru conversia formatelor existente în FBX. Convertirea unui fișier FBX într-un alt format însă se realizează destul de complicat și poate fi însoțită de apariția erorilor.

Un fișier OBJ va fi, mult mai simplu și de dimensiuni reduse comparativ cu un fișier FBX sau COLLADA care conțin același model 3D. Acest lucru se datorează simplității formatului de fișier OBJ în comparație cu celelalte specificații și datorită codificării sale binare native.

Astfel, dacă nu este nevoie de o scenă complexă sau de animații și dar mult mai mult contează suportul și interpretare corectă de diferite software CAD, formatul de fișier OBJ este formatul potrivit. În aproape toate celelalte cazuri, FBX este formatul optim pentru aplicațiile grafice 3D.

### Caracteristici moderne a formatului de fișiere obj

În ceea ce privește caracteristicile moderne, formatul FBX este cel mai progresiv format care oferă o mulțime de caracteristici de ultimă generație, actualizări și îmbunătățiri regulate. Formatul de fișier OBJ este al doilea în ceea ce privește caracteristicile, în timp ce formatul COLLADA nu se bucură de actualizări și îmbunătățiri regulate.

## Geometrie

Scopul principal al formatului de fișier OBJ este de a codifica geometria suprafeței unui obiect 3D. Formatul de fișier OBJ este destul de versatil în acest sens. Permite mai multe opțiuni pentru codarea geometriei suprafeței. Fiecare dintre cele trei metode permise descrise mai jos au propriile avantaje și dezavantaje.

### Teselare cu fețe poligonale

În forma sa cea mai simplă, formatul de fișier OBJ permite utilizatorului să seleze suprafața modelului 3D cu forme geometrice simple precum triunghiuri, patrulatere sau poligoane mai complexe. Vârfurile poligoanelor și normala fiecărui poligon sunt apoi stocate într-un fișier care conține geometria suprafeței modelului.

Teselările cu fețe poligonale au o serie de avantaje și dezavantaje. Poligoanele sunt forme geometrice simple și această metodă este de fapt cel mai simplu mod de a descrie geometria suprafeței. Cu toate acestea, aproximarea unei suprafețe curbă cu poligoane duce la modificarea grosimii modelului.

În cazul tipăririi 3D, imprimanta 3D va imprima obiectul cu aceeași grosime specificată de fișier. Desigur, făcând triunghiurile din ce în ce mai mici, aproximarea poate fi făcută din ce în ce mai exact, rezultând imprimări de o calitate mare. Cu toate acestea, pe măsură ce se micșorează dimensiunea triunghiului, crește și numărul de triunghiuri necesare pentru a descrie suprafața. Acest fapt cauzează creșterea semnificativă a dimensiunii fișierului, problemă care încearcă să o soluționeze slicer-ele de imprimare 3D. De asemenea, devine dificilă distribuția sau gestionarea unor astfel de fișiere uriașe.

Prin urmare, este foarte important să se găsească echilibrul corect între dimensiunea fișierului și calitatea imprimării. Nu are sens să se reducă dimensiunile triunghiurilor la infinit, deoarece la un moment dat nu se va mai putea distinge cu ochiul liber diferența între calitățile tipăririi.

Extensia de fișier .obj se referă în primul rând la Wavefront 3D (.obj) dezvoltat de Wavefront Technologies pentru software-ul Advanced Visualizer. Formatul OBJ este un format textual pentru descrierea geometriei corpurilor tridimensionale care permite modelarea formelor volumetrice complexe și aplicarea diverselor materiale și texturi. Pe lângă fișierul .OBJ, un obiect sau scenă tipică Wavefront 3D va include, de obicei, unul sau mai multe fișiere Material Template Library (.mtl), care definesc materialele obiectului cu referințe la texturi de tip bitmap externe, de obicei stocate într-un subdirector separat.

Formatul OBJ a devenit unul dintre cele mai populare și acceptate formate de modele 3D și funcțiile de export/import de fișiere .OBJ sunt prezente în aproape fiecare editor 3D. Destul de multe utilizare pentru vizualizarea modelelor 3D (există chiar și aplicații web care permit importul, vizualizarea, editarea, exportul și conversia fișierelor .OBJ) oferă posibilitatea de a deschide fișiere .OBJ și a afișa modelele conținute cu redare completă, iar o serie de convertori permit conversia modelelor OBJ în alte formate. Există colecții întregi și biblioteci de modele în acest format pe Internet. Formatul de geometrie OBJ este un format deschis pentru fișierelor de descriere a geometriei.

### Specificație OBJ

Formatul de fișier OBJ este un format de fișier ASCII. Editarea acestuia poate fi realizată cu ajutorul oricărui editor de text.

Specificația originală nu indică explicit care ar trebui să fie caracterul de sfârșit de linie, așa că unele software-uri folosesc \r return de car (CR) și altele folosesc \n linie nouă (NL sau LF). Este posibil să fie necesară conversia caracterelor care indică sfârșitul de linie dacă editorul dvs. de text sau software-ul utilizat nu reușesc să citească fișierul.

Primul caracter al fiecărei linii este foarte important deoarece acesta specifică tipul de comandă. Dacă primul caracter este #, acea linie reprezintă un comentariu și orice altceva din acea linie este ignorat. Liniile goale sunt, de asemenea, ignore.

### Comanda comentariu

# o linie de comentarii

După cum s-a menționat anterior, caracterul # indică faptul că linia reprezintă un comentariu și ar trebui ignorată. Prima linie este de obicei întotdeauna un comentariu care specifică ce program a generat fișierul.

### Comanda vîrf

v x y z

În această versiune simplificată a specificației, vor fi cuprinse doar fețele poligonale, prin urmare comanda **vertex** – v poate fi utilizată pentru a specifica vârfurile poligonului. Când se utilizează suprafețe și curbe de formă liberă, există o comandă similară **vp** care poate fi utilizată pentru a specifica punctele de control ale suprafeței sau curbei.

Comanda **vertex** – v specifică un vârf prin cele trei coordonate carteziene **x**, **y** și **z**. Vârfului *i* se atribuie automat un nume în funcție de ordinea în care este găsit în fișier. Primul vârf din fișier primește numele „1”, al doilea ca „2”, al treilea ca „3” și aşa mai departe.

### Comanda normală la vîrf

vn x y z

**Vertex normal** – **vn** este comanda pentru normala unui vârf. Specifică vectorul normal la suprafață. Parametrii **x**, **y** și **z** sunt componentele vectorului normal. Acest vector normal nu este încă asociat niciunui vârf. Acesta va trebui să fie asociat ulterior unui vârf cu ajutorul unei alte comenzi numită comanda **f**.

Comanda vertex normal poate fi omisă într-o mulțime de fișiere, deoarece atunci când se vor grupa vârfurile în fețe poligonale cu comanda **f**, se va determina automat vectorul normal din coordonatele vârfului și ordinea în care vor apărea vârfurile.

### Comanda textură a vîrfului

vt u v [w]

Comanda **vertex texture** – **vt** specifică un punct din harta texturilor. Parametrii **u** și **v** sunt coordonatele **x** și **y** din harta texturii. Acestea vor fi numere în virgulă mobilă între 0 și 1. Acestea trebuie asociate unui vârf cu ajutorul comenzi **f**, analogic normalelor la vârfuri.

### Comanda suprafață

f v1 [/ vt1] [/ vn1] v2 [/ vt2] [/ vn2] v3 [/ vt3] [/ vn3] ...

Comanda **suprafață** – **f** este probabil cea mai importantă comandă. Specifică o suprafață poligonală creată din vârfurile specificate anterior.

Pentru a face referire la un vârf, trebuie doar urmat sistemul de numerotare implicit al vârfurilor. De exemplu, „f 23 24 25 27” descrie o față poligonală construită din vârfurile 23, 24, 25 și 27 în ordine.

Pentru fiecare vârf, se poate asocia o comandă **vn**, care apoi asociază acel normală cu vârful corespunzător. În mod similar, poate fi asociată o comandă **vt** cu un vârf, care va determina maparea texturii utilizate pentru acest vârf.

Dacă se indică textura sau normala pentru un vârf, atunci acestea trebuie specificate pentru toate.

## Comanda grup

g nume

Comanda **grup** – **g** indică gruparea unei părți a obiectului într-un grup cu numele indicat ca parametru. Toate comenziile **f** care urmează vor fi în incluse în același grup. Acest lucru este util dacă se dorește reutilizarea unei anumite informații, cum ar fi tipul de material, pentru o parte selectată a unui obiect.

## Comanda material utilizat

nume usemtl

Comanda de **utilizare a materialului** – **usemtl** permite indicarea materialului utilizat. Toate comenziile **f** care vor urma vor folosi același material până nu apare o altă comandă **usemtl**.

## 3.2 Biblioteca de materiale

Informația despre aspectul obiectelor (materialelor) conținute în fișierul OBJ se conține în fișiere separate în format MTL (Material Library). Fișierul OBJ face referință la un astfel de fișier, dacă este necesar, folosind directiva:

Mtllib [numele fișierului extern MTL]

MTL este un standard stabilit de Wavefront Technologies. Toate informațiile sunt prezentate în formă ASCII și sunt lizibile pentru om. Standardul MTL este, de asemenea, foarte popular și este acceptat de majoritatea pachetelor grafice 3D.

Informațiile despre materialele simple din fișier arată astfel:

```
Newmtl material_name1
# Anunțarea următorului material
# Culori Ka 1.000 1.000 0.000
# Culoare lumină ambientală (galben) Kd 1.000 1.000 1.000
# Culoare difuză (alb)
# Parametri de reflecție Ks 0.000 0.000 0.000
# Culoare reflexie speculară (0; 0; 0 - oprit) Ns 10.000
# Coeficient de reflecție speculară (de la 0 la 1000)
# Parametri de transparență d 0.9
# Transparență este specificată folosind directiva d Tr 0.9
# sau în alte implementări de format folosind Tr
# Următorul material newmtl material_name2 ...
```

Toți parametrii sunt optionali. În absența oricărui parametru, programul îl setează automat în mod implicit.

Având în vedere toate cele expuse anterior, mai jos este prezentat conținutul unui fișier OBJ simplu (CUB) construit în editorul 3D Google SketchUp, Fig 3.1 și exportat ca fișier .OBJ cu ajutorul plugin-ului *LIPID Lightsolve* care poate fi ușor găsit și instalat din meniul

Window>Extension Warehouse. Pentru a avea posibilitate de a instala oricare plugin din biblioteca de extensii utilizatorul trebuie să dispună de un account Google și să fie autentificat.

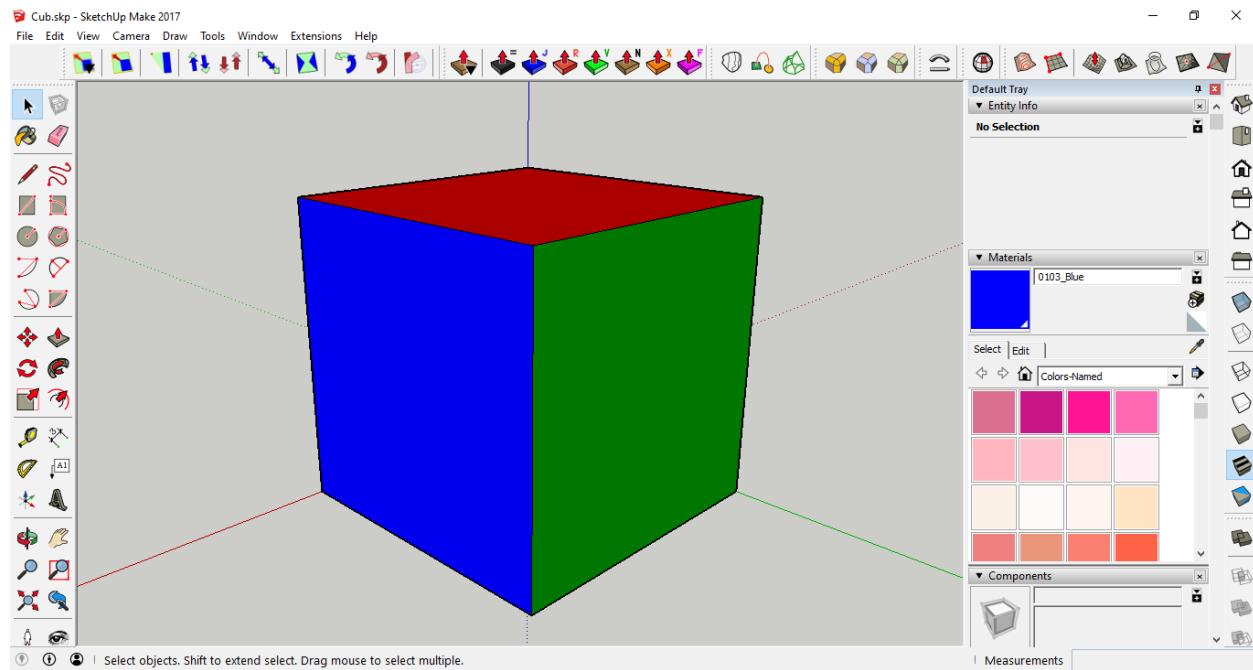


Fig. 3.1 Modelul 3D a unui cub construit în editorul 3D Google SketchUp.

Pentru a exporta modelul 3D al cubului în formatul OBJ se accesează: File-> LIPID OBJ Exporter, Fig3.2, apoi în fereastra LIPIDOBJ se bifează optiunea Ignore Back face.

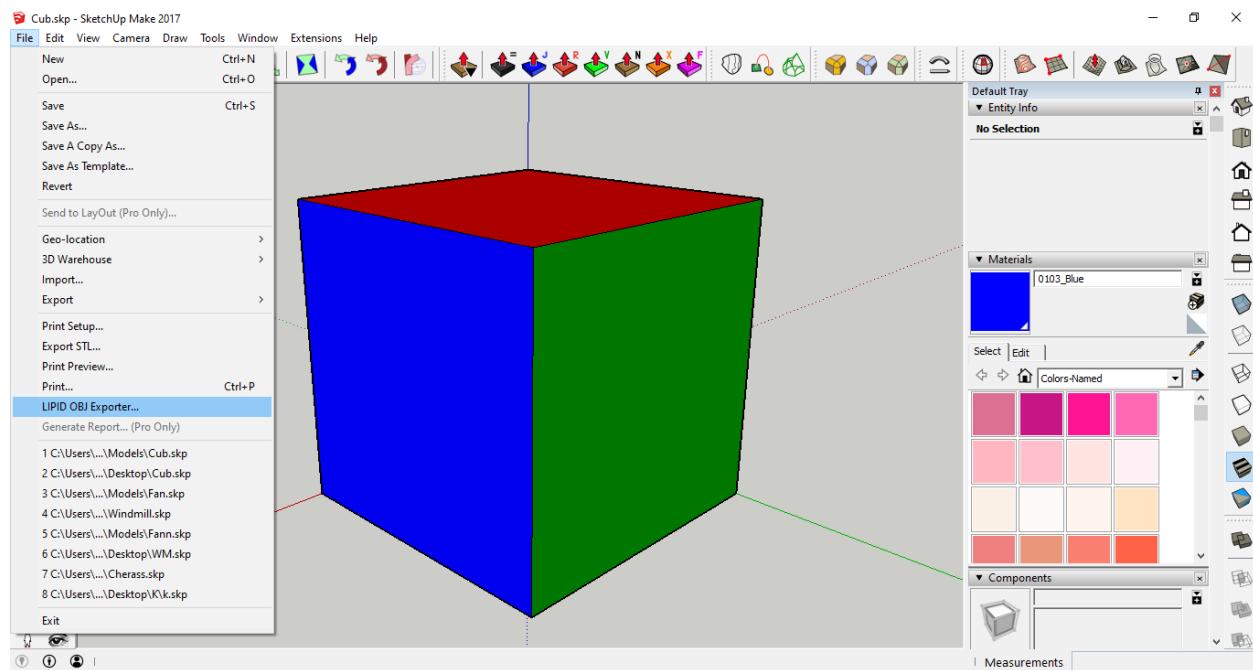


Fig. 3.2 Exportarea modelul 3D al cubului în formatul OBJ cu ajutorul plugin-ului LIPIDOBJ.

Conținutul fișierului .OBJ a modelului 3D a cubului prezentat în Fig. 3.2:

*mtllib /Cub.mtl*

*s 1*

*o*

```

g
usemtl 0020_Red
v 100 100 0
v 0 0 0
v 0 100 0
vn 0 0 -1
f 1//1 2//1 3//1
v 100 0 0
f 2//1 1//1 4//1
v 100 0 100
v 0 100 100
v 0 0 100
vn 0 0 1
f 5//2 6//2 7//2
v 100 100 100
f 6//2 5//2 8//2
usemtl 0076_Green
vn 0 -1 0
f 5//3 2//3 4//3
f 2//3 5//3 7//3
vn 0 1 0
f 6//4 1//4 3//4
f 1//4 6//4 8//4
usemtl 0103_Blue
vn -1 0 0
f 6//5 2//5 7//5
f 2//5 6//5 3//5
vn 1 0 0
f 1//6 5//6 4//6
f 5//6 1//6 8//6

```

Analizînd conținutul fișierului .OBJ prezentat anterior se poate observa faptul că în fișierul Cub.obj este utilizat fișerul de materiale Cub.mtl care se află în același. Pe lîngă aceasta mai poate fi observată descrierea coordonatelor tuturor celor 8 vîrfuri ale cubului, indicarea suprafețelor (triunghiurilor) care alcătuiesc cele 6 fețe ale cubului cu specificarea normalelor acestora și atribuirea texturilor 0020\_Red, 0076\_Green și 0103\_Blue suprafețelor (triunghiurilor) corespunzătoare.

Conținutul fișierului materialelor utilizate pentru texturarea modelului 3D a cubului prezentat în Fig. 3.2.

```
#  
## Alias MTL Material File  
# Converted from SKP by LIPID Lightsolve  
newmtl 0020_Red  
Ka 0.000000 0.000000 0.000000  
Kd 1 0 0  
Ks 0.330000 0.330000 0.330000  
newmtl 0076_Green  
Ka 0.000000 0.000000 0.000000  
Kd 0 0.501961 0  
Ks 0.330000 0.330000 0.330000  
newmtl 0103_Blue  
Ka 0.000000 0.000000 0.000000  
Kd 0 0 1  
Ks 0.330000 0.330000 0.330000  
newmtl default_material  
Ka 0.000000 0.000000 0.000000  
Kd 0.330000 0.330000 0.330000  
Ks 0.330000 0.330000 0.330000
```

Analizând conținutul fișierului .MTL prezentat anterior poate fi observată definirea texturilor *0020\_Red*, *0076\_Green* și *0103\_Blue* utilizate în fișierul Cub.obj pentru texturarea fețelor modelului cubului.

### **Descrierea obiectelor și metodelor principale pentru lucrul cu fișiere .OBJ cu ajutorul librăriei pentru realizarea grafică WEB treidimensionale în editorul online p5.js.**

Biblioteca p5.js este o bibliotecă JavaScript care ușurează procesul de dezvoltare a programelor ce utilizează elemente grafice 2D și 3D și este destinată atât pentru dezvoltatorii experimentați cât și pentru începători. Biblioteca p5.js este gratuită și open-source.

Biblioteca p5.js are un set complet de funcționalități grafice. Cu ajutorul acestei biblioteci grafice întreaga pagină a browserului este privită ca spațiu de lucru în care pot fi utilizate inclusiv obiecte HTML5 pentru text, video, cameră web și sunet.

#### **Funcțiile de bază:**

Încărcarea unui model 3D dintr-un fișier OBJ sau STL.

*loadModel()* trebuie plasat în interiorul funcției *preload()*. Acest lucru permite modelului să se încarce complet înainte de a rula restul programului.

Una dintre limitările formatului OBJ și STL este faptul că nu ține cont de scară. Aceasta înseamnă că modelele exportate din diferite programe pot avea dimensiuni diferite. Dacă modelul nu se afișează, încercați să apelați ***loadModel()*** cu parametrul de normalizare setat la true. Aceasta va redimensiona modelul la o scară adecvată pentru p5. De asemenea, se pot face modificări suplimentare ale dimensiunii modelului cu funcția ***scale()***.

De asemenea, nu este realizat suportul pentru fișiere colorate STL. Fișierele STL colorate vor fi redate fără proprietăți de culoare.

#### **Sintaxa:**

`loadModel (path, normalize, [successCallback], [failureCallback], [fileType])`

`loadModel (path, [successCallback], [failureCallback], [fileType])`

#### **Parametri:**

path String: Calea modelului încărcat

normalize boolean: dacă este adevărat, se scalează modelul la o dimensiune standardă la încărcare

funcția successCallback Function(p5.Geometry): funcție care trebuie apelată după încărcarea modelului. Va fi returnat obiectul de tip 3D model. (Optional)

funcția failureCallback Function(Event): apelat cu eveniment de eroare dacă modelul nu reușește să se încarce. (Optional)

fileType String: extensia fișierului modelului (.stl, .obj). (Optional)

Se întoarce

p5.Geometry: obiect de tip p5.Geometry

#### ***model()***

Redă un model 3D pe ecran.

#### **Sintaxa:**

`model(model)`

#### **Parametri:**

model p5.Geometry: Model 3D încărcat care trebuie redat

Sketch Files->Upload file

```

let fr = 30;
function preload() {
  cub = loadModel('Cub.obj');
}
function setup() {
  createCanvas(600, 500, WEBGL);
  normalMaterial();
  frameRate(fr);
}
function draw() {
}

```

Fig. 3.3 Încarcarea fișierului Cub.obj în directoriul proiectului.

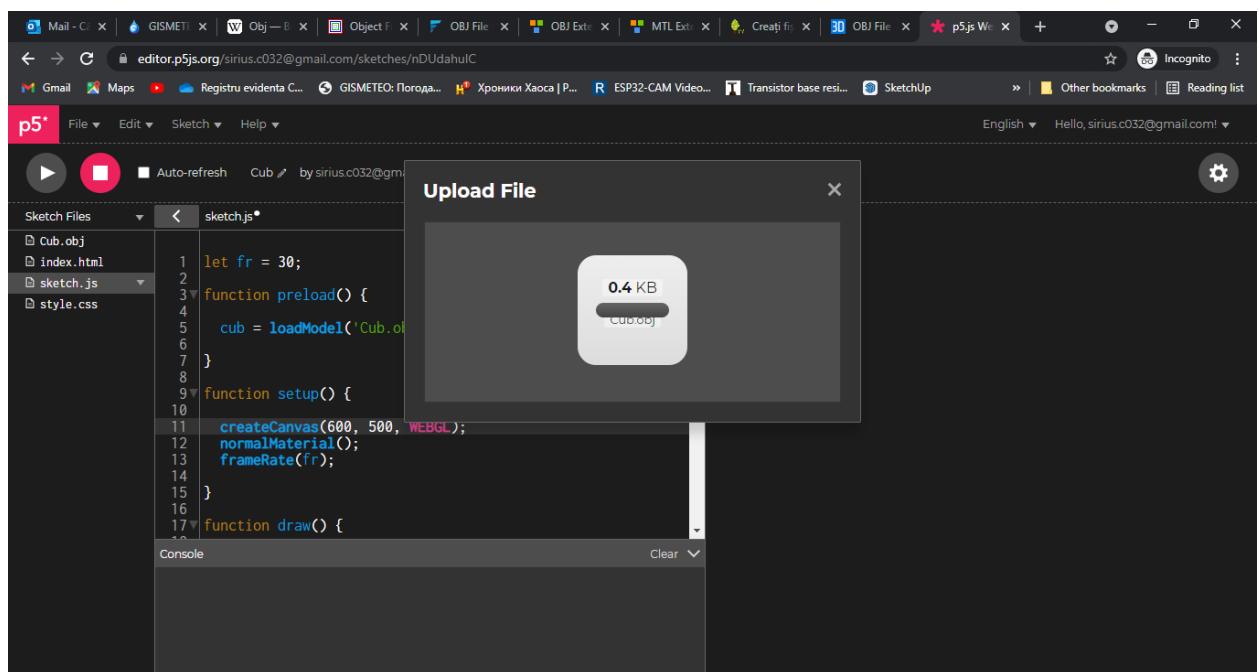


Fig. 3.5 Rezultatul procesului de încarcare a fișierului Cub.obj în directoriul proiectului.

```

let fr = 30;
function preload()
{

```

```

cub = loadModel('Cub.obj');

}

function setup()
{

    createCanvas(600, 500, WEBGL);
    normalMaterial();
    frameRate(fr);
}

function draw()
{
    orbitControl();
    background(50);
    normalMaterial();
    scale(1.0);
    stroke(100, 100, 100);
    strokeWeight(0.3);
    fill(150, 150, 150);
    model(cub);
}

```

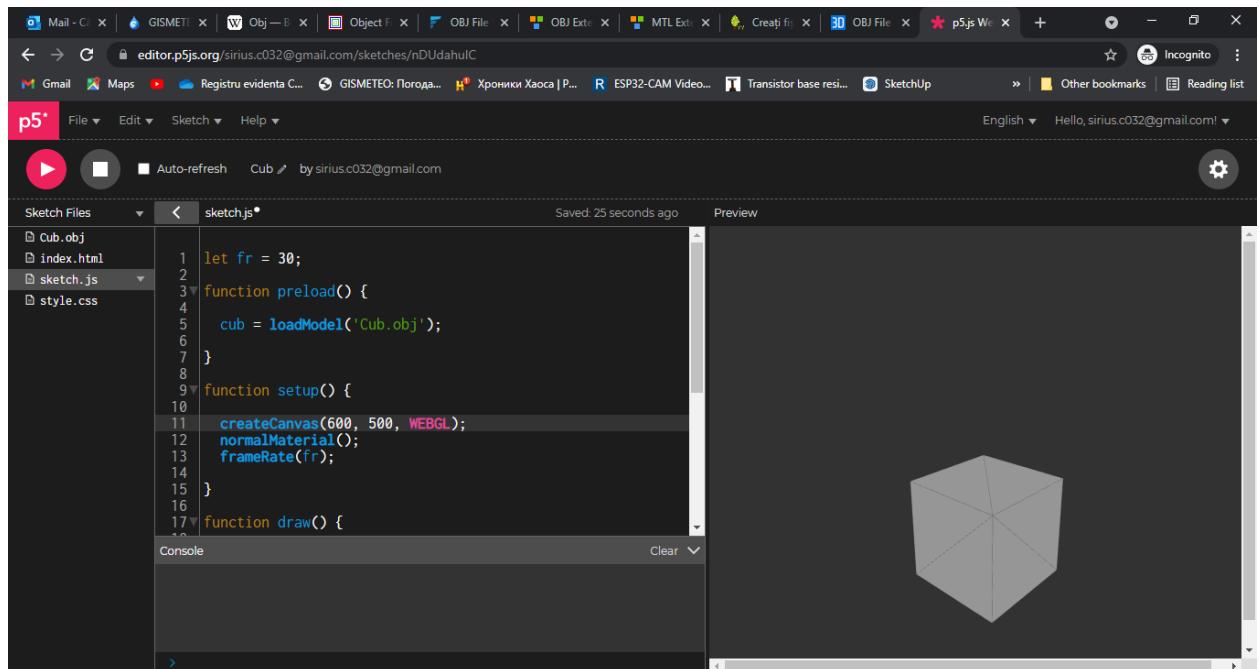


Fig. 3.5 Rezultatul execuției programului de afișare a modelului 3D a cubului.

## Crearea unei scene statice 3D.

Sarcina lucrării de laborator constă în conceperea și construirea unei scene statice 3D cu ajutorul editorului online p5.js utilizând modele de obiecte 3D stocate în fișiere .OBJ create individual sau descărcate din internet.

Pentru exemplificarea mersului lucrării de laborator nr. 3 se pune ca scop crearea unei scene statice care ar reprezenta o moară de vînt îngrădită de un gard în ograda căreia se vor afla cîteva animale deomestice, un arbore și un stog de fin. Modelele 3D ale acestor obiecte pot fi create de la zero sau importate din repozitoriu 3D Warehouse care poate fi accesat din meniu de bază al editorului grafic 3D Google SketchUp *Window->3D Warehouse*, apoi editate după necesitate.

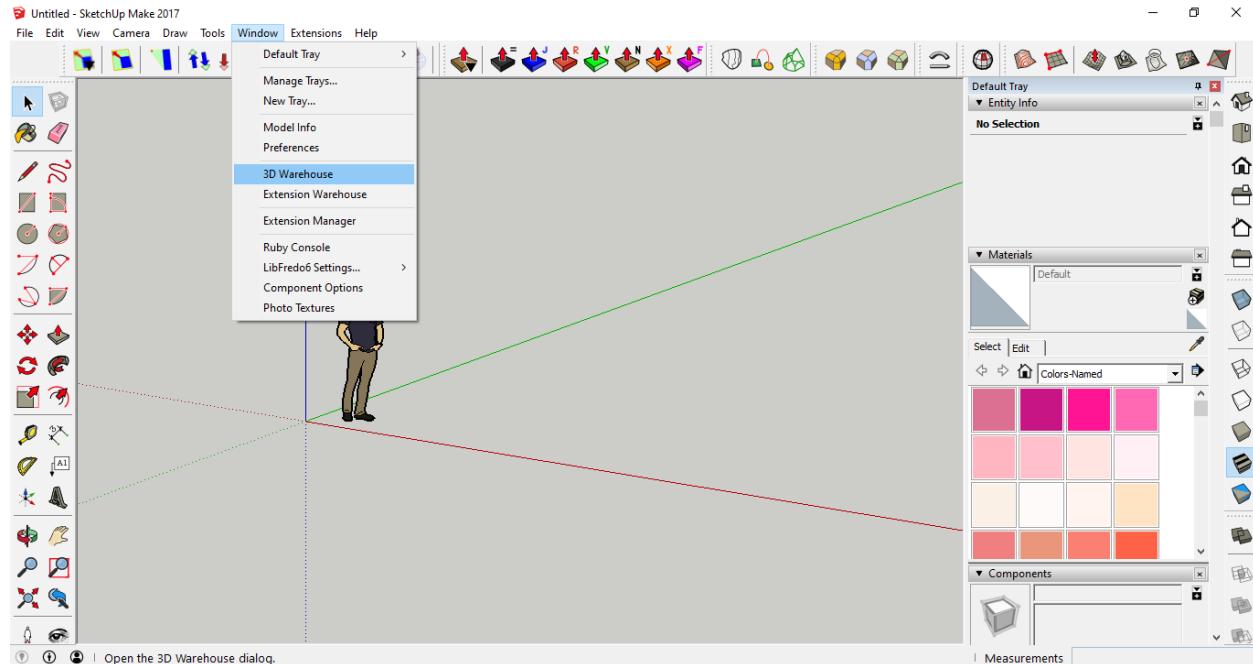


Fig. 3.6 Accesarea repozitoriului 3D Warehouse.

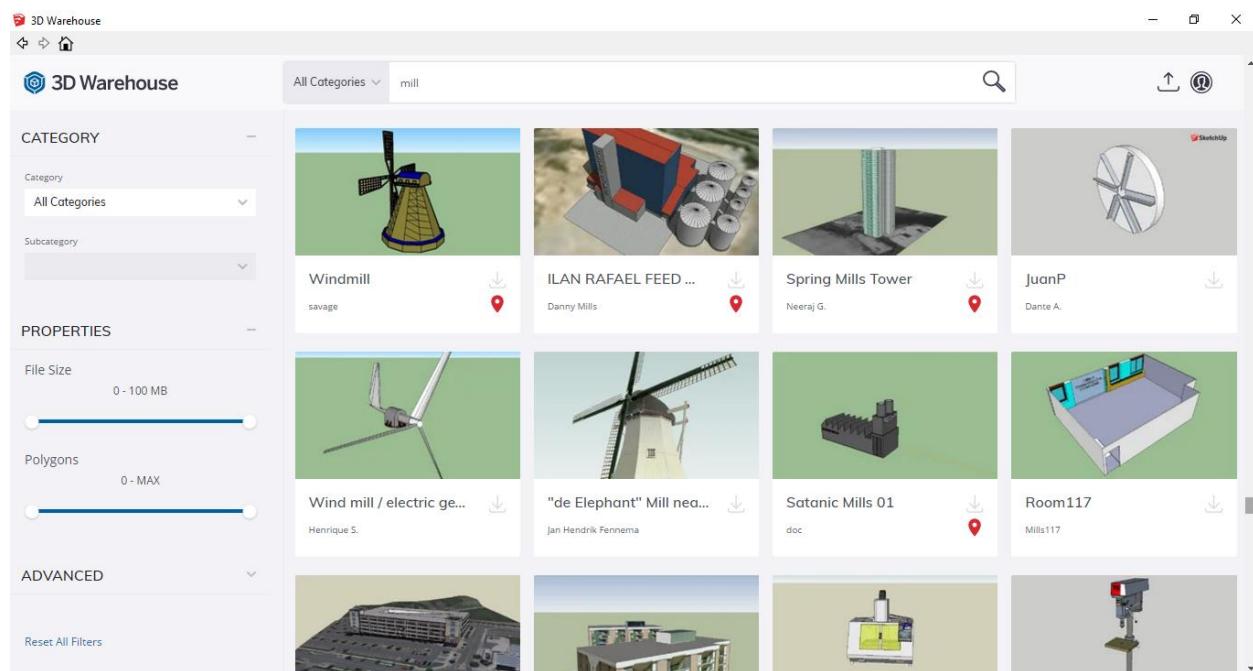


Fig. 3.7 Căutarea modelelor 3D în repozitoriul 3D Warehouse.

Pentru a avea posibilitatea de a downloada sau importa modelul selectat în sketch-ul curent utilizatorul trebuie să dispună de un cont Google Account și să fie autentificat în mediul 3D Warehouse. După ce a fost downloadat modelul/modelele necesare, acestea pot fi editate după necesitate și exportate ca fișiere .OBJ prin intermediului plug-inului *LIPID OBJ Exporter* care poate fi instalat din meniul *Window->Extension Warehouse*, Fig. 3.7, în cîmpul de căutare al căruia indicăm cuvîntul cheie OBJ iar ca răspuns sînt afișate plugin-urile care satisfac criteriile de căutare printre care poate fi observat și plugin-ul căutat *LIPID OBJ Exporter*, Fig. 3.9. Pentru a avea posibilitatea de a instala plugin-urile, utilizatorul trebuie să dispună de un cont Google Account și să fie autentificat în repozitoriu online *Extension Warehouse*.

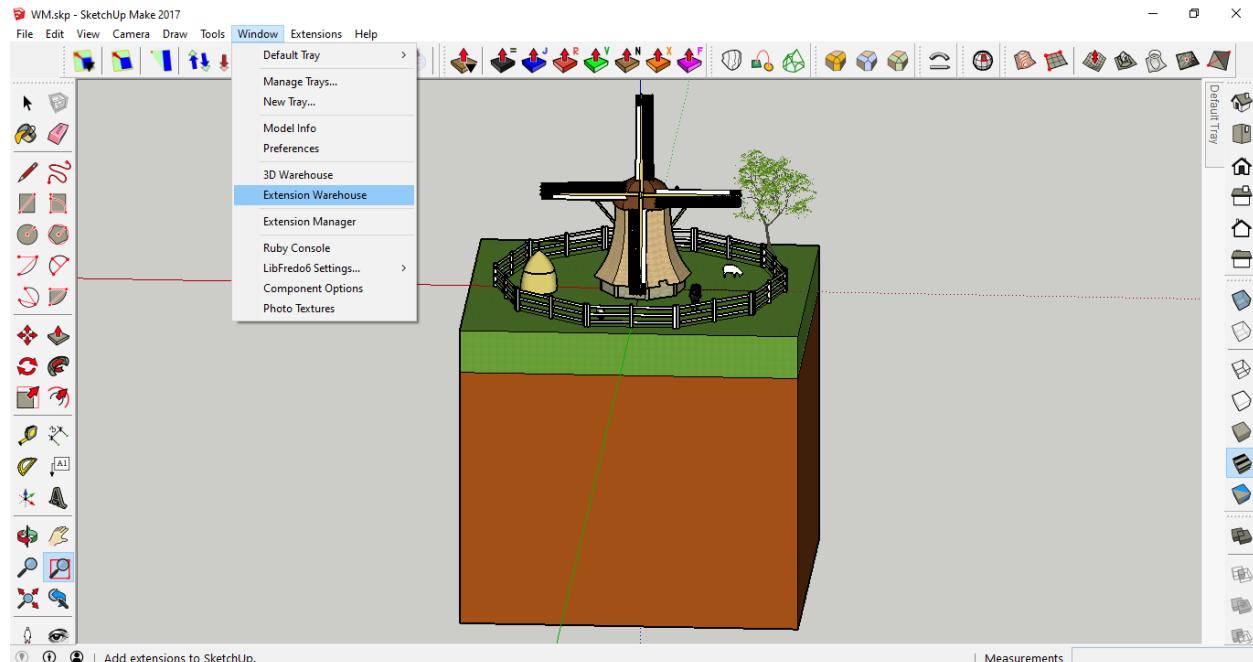


Fig. 3.8 Accesarea opțiunii *Extension Warehouse* din bara de meniu.

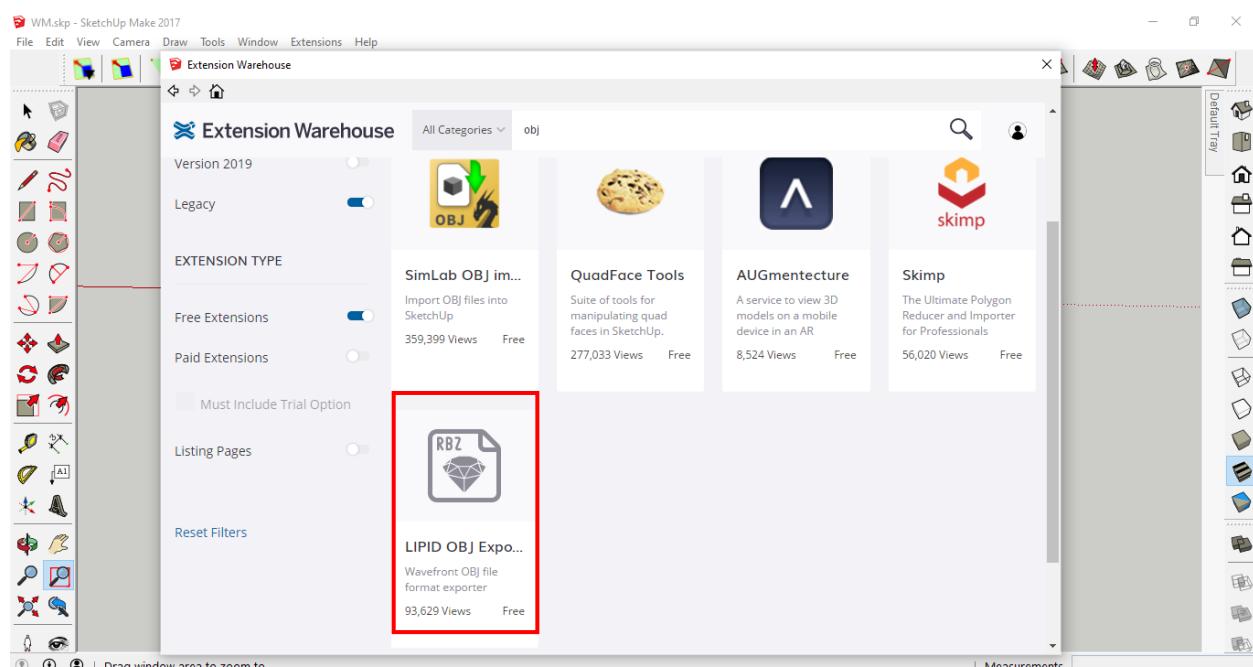


Fig. 3.9 Fereastra de căutare și instalare a plugin-urilor, din repozitoriu online *Extension Warehouse*.

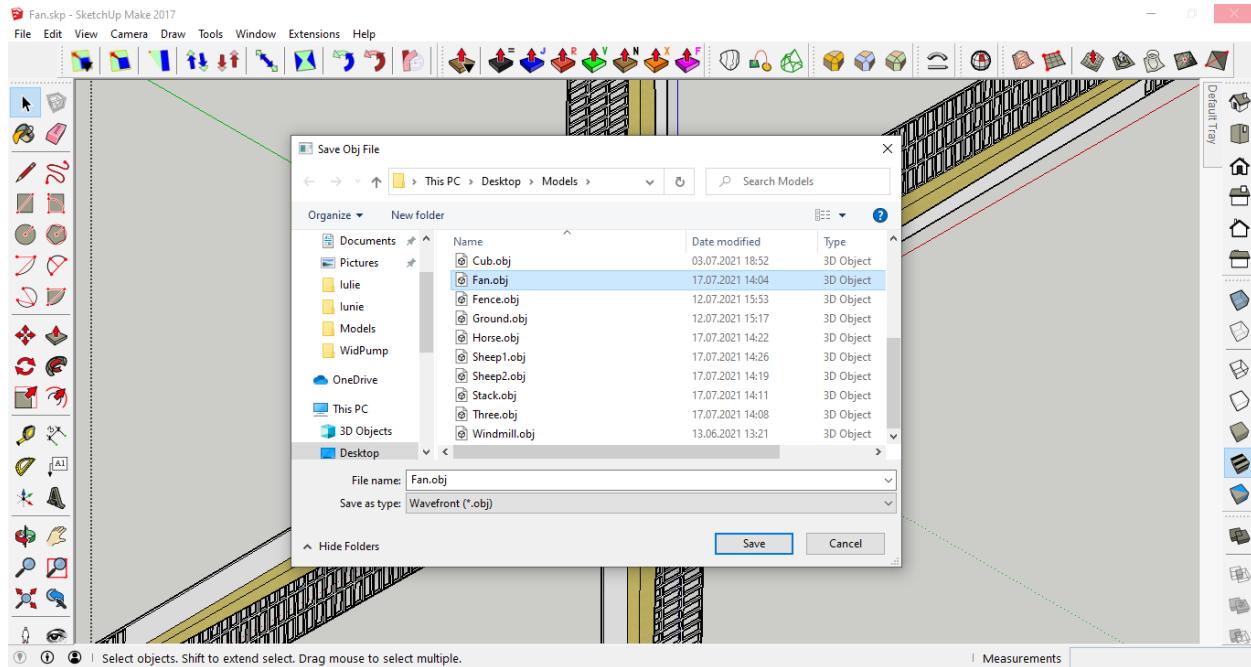


Fig. 3.10 Fereastra de salvare în format .obj a modelelor 3D a obiectelor din scenă.

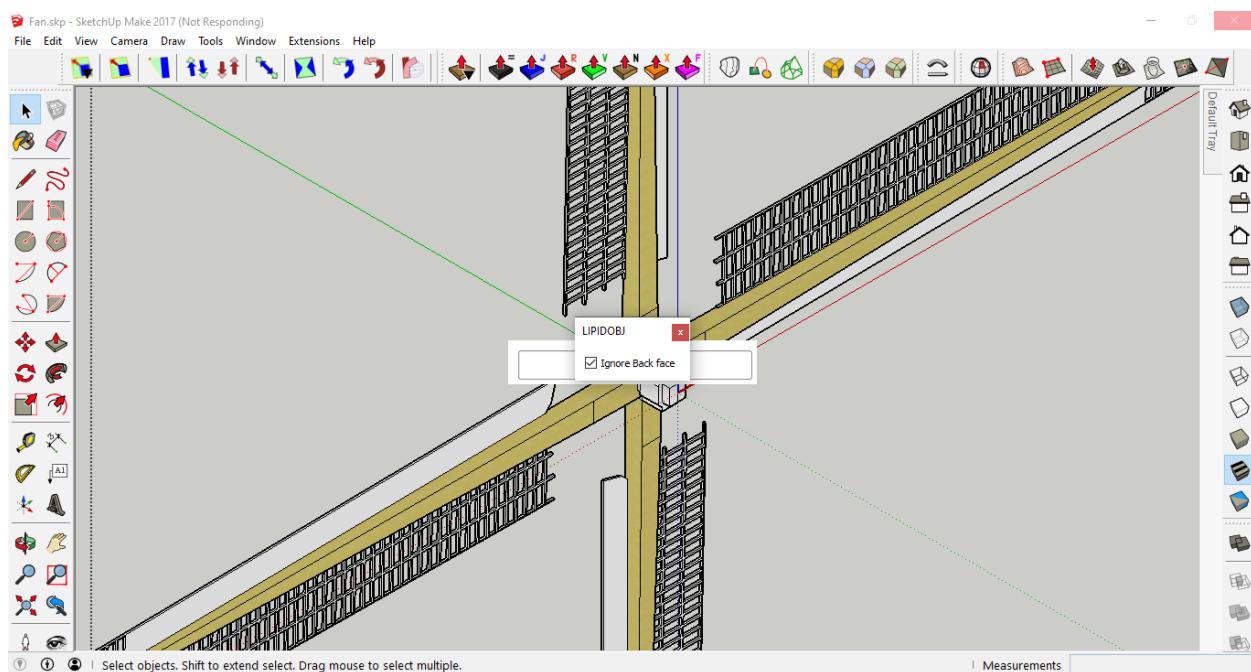


Fig. 3.11 Opțiunea de ignorare a fețelor a plugin-ului LIPIDOBJ la exportarea modelelor 3D în format .obj.



Fig. 3.12 Vizualizarea conținutului fișierelor .OBJ cu ajutorul aplicației standard din cadrul sistemului de operare Windows 10 – 3D Viewer.

```

let fr = 30;
let obj;

function preload() {

    mill = loadModel('Windmill.obj');
    fan = loadModel('Fan.obj');
    gnd = loadModel('Ground.obj');
    sheep1 = loadModel('Sheep1.obj');
    sheep2 = loadModel('Sheep2.obj');
    horse = loadModel('Horse.obj');
    stack = loadModel('Stack.obj');
    fence = loadModel('Fence.obj');
    three = loadModel('Three.obj');

}
  
```

```

function setup() {

    createCanvas(400, 400, WEBGL);
    normalMaterial();
  
```

```
//ambientMaterial(150, 150, 150);
frameRate(fr);
angleMode(DEGREES);
}

function draw() {
    orbitControl();
    background(50);
    pointLight(255, 255, 255, 100000, 100000, -100000);
    noStroke();

    push();
    scale(0.005);
    translate(0, 0, 0);
    model(mill);
    pop();

    push();
    scale(0.005);
    translate(0, 3000, 13000);
    rotateX(15);
    model(fan);
    pop();

    push();
    scale(0.005);
    model(gnd);
    pop();

    push();
    scale(0.005);
    translate(-3000, 15000, 0);
    rotateZ(-10);
    model(sheep1);
    pop();
}
```

```
push();
scale(0.005);
translate(10000, 0, 0);
rotateZ(180);
model(sheep2);
pop();

push();
scale(0.005);
rotateZ(-15);
translate(7000, 12000, 0);
model(horse);
pop();

push();
scale(0.005);
translate(-12000, -2000, 0);
model(stack);
pop();

push();
scale(0.005);
translate(0, 0, 0);
model(fence);
pop();

push();
scale(0.005);
translate(15000, -15000, 0);
model(three);
pop();
}
```

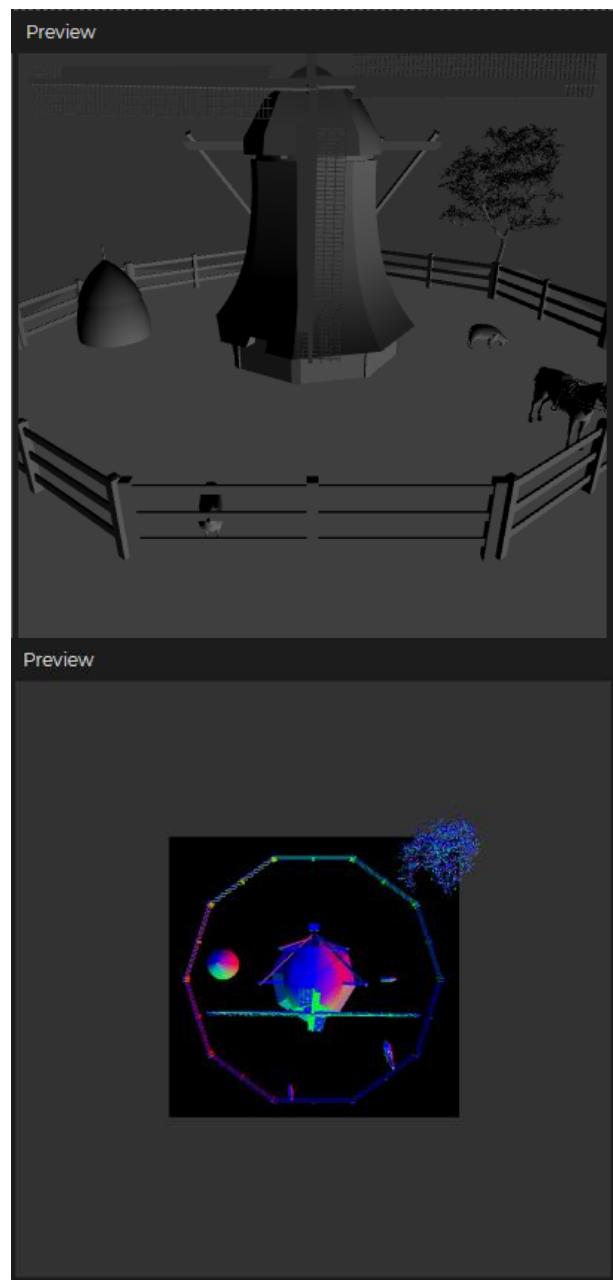
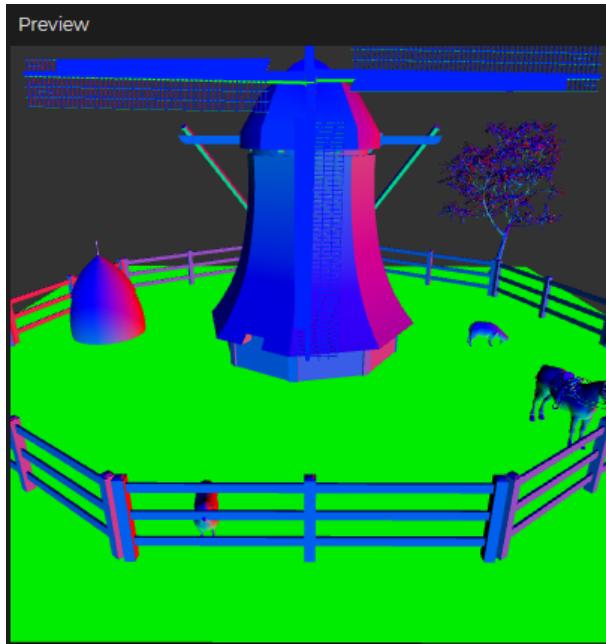


Fig. 3.13 Rezultatul execuției programului de afișare a scenei statice.



## **Lucrarea de laborator nr. 3**

### **Tema: Scene statice 3D**

Scopul lucrării: Obținerea cunoștințelor practice în sinteza scenelor grafice 3D statice, utilizând funcțiile standard de reprezentare a modelelor 3D din biblioteca p5.js.

Sarcina lucrării:

1. Elaborați un program pentru sinteza unei scene 3D statice care ar conține cel puțin 5 obiecte utilizând funcțiile standard de reprezentare a modelelor 3D din biblioteca p5.js.
2. Elaborați un program care crează o scenă 3D statică conform variantei indicate în tabelul 3.1. Pentru crearea scenei pot fi utilizate obiecte grafice 3D existente în repositoriu 3D Warehouse.

**Tabelul 3.1 Variantele pentru realizarea lucrării de laborator**

N r	Obiect 3D	Model	N r	Obiect 3D	Model
1	Carusel		6	Morișcă de vînt	

2	Far maritim		7	Turbină eoliană	
3	Automobil blindat		8	Elicopter	
4	Moară de apă		9	Submarină	
5	Avion cu elice		10	Catapultă	

## Transformari 3D

### Sistem de coordonate

În geometrie, un sistem de coordonate reprezintă o modalitate prin care oricărui punct i se asociază în mod unic o mulțime ordonată de numere reale, numite coordonatele aceluui punct. În spațiul euclidian sunt necesare trei coordonate (abscisa, ordinata și cota), în plan sunt necesare două (abscisa și ordinata), iar pentru localizarea punctelor pe o dreaptă este necesară doar o coordonată. În geometria analitică, utilizarea sistemelor de coordonate permite transformarea problemelor de geometrie în probleme de algebră.

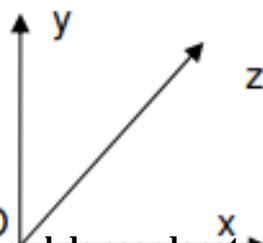
Vom utiliza coordonate omogene ca și în cazul 2D

În matematică, coordonatele omogene, introduse de August Ferdinand Möbius, permit transformări affine prin reprezentarea lor sub forma unei matrici. Coordonatele omogene permit, de asemenea, efectuarea calculelor în spații proiective într-un mod similar cu cel în care coordonatele carteziene o fac în spațiul euclidian.

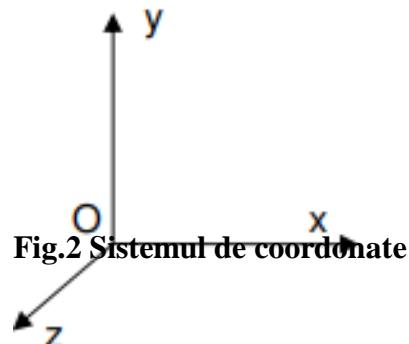
Coordonatele omogene ale unui punct din spațiu proiectiv de dimensiune  $n$  sunt de obicei scrise ca  $(x: y: z: \dots: w)$ , un vector linie de lungime  $n + 1$ , altele decât  $(0: 0: 0: \dots: 0)$ . Două seturi de coordonate, care sunt proporționale denotă același punct din spațiu proiectiv: pentru orice non-zero scalar din domeniu care stă la baza  $K$ ,  $(cx : cy : cz : \dots : cw)$  reprezintă același punct. Prin urmare, acest sistem de coordonate poate fi explicitat după cum urmează: în cazul în care spațiu proiectiv este construit dintr-un spațiu vectorial  $V$  de dimensiune  $n + 1$ , se introduc coordonatele în  $V$ , prin alegerea unei baze, și utilizarea acestora în  $P(V)$ , clasele de echivalență proporționale non-zero vectori în  $V$ .

Transformările vor fi prezentate prin matrice  $4 \times 4$

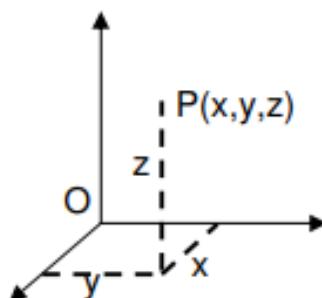
Vom utiliza sistemul de coordonate de dreapta (right-handed)



**Fig.1 Sistemul de coordonate orientat stînga. orientat dreapta.**



**Fig.2 Sistemul de coordonate**



**Fig.3 Sistem de coordonate carteziene.**

### Transformări 3D de bază

#### Translația

$$T(t_x, t_y, t_z) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

#### Rotatia în jurul unei axe

$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

*Scalarea*

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### Transformări 3D compuse

*Componerea transformărilor*

*Matricea generalizată*

Structura pentru cazul notației prin vector-coloana:

- matricea 3x3 include transformări de scalare locală, forfecare, oglindire și rotație
- matricea 3x1 reprezintă transformarea de translație
- matricea 1x3 reprezintă transformarea de proiecție perspectivă
- matricea 1x1 reprezintă transformarea de scalare generală

$$\begin{bmatrix} & & & : & 3 \\ & 3 \times 3 & & : & \times \\ & & & : & 1 \\ \dots & \dots & \dots & \dots & \dots \\ 3 & \times & 1 & : & 1 \times 1 \end{bmatrix}$$

Matricea corespunzătoare transformării compuse se obține prin înmulțirea matricelor transformărilor elementare. Deoarece înmulțirea matricelor nu este comutativă, este importantă ordinea în care se aplică aceste transformări

Matricea de transformare cea mai apropiată vectorului linie (sau a vectorului coloană) corespunde primei transformări care se aplică în timp ce matricea de transformare cea mai depărtată este ultima dintre cele aplicate

Matematic aceasta se exprimă prin:

$$[M] [VC] = [M_1] \dots [M_3] [M_2] [M_1] \dots [VC]$$

pentru vector coloana

sau

$$[VL] [M] = [VL] [M_1]^T [M_2]^T [M_3]^T \dots [M_n]^T$$

pentru vector linie unde  $[M_i]$  poate fi orice matrice de transformare

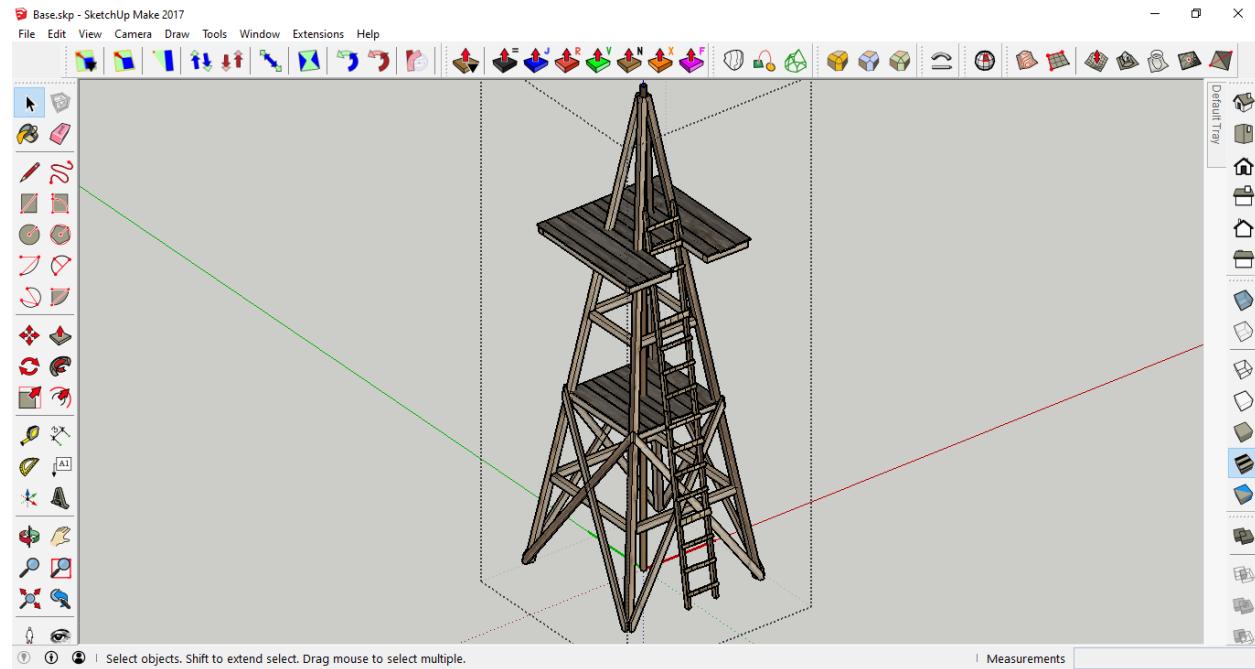
Scopul lucrării: Crearea unei scene dinamice 3D.

Sarcina lucrării de laborator constă în conceperea și construirea unei scene dinamice 3D care ar conține transformările: translație, rotație și scalare. Realizarea scenei cu ajutorul editorului online p5.js utilizând modele de obiecte 3D stocate în fișiere .OBJ care pot fi create individual sau descărcate din internet.

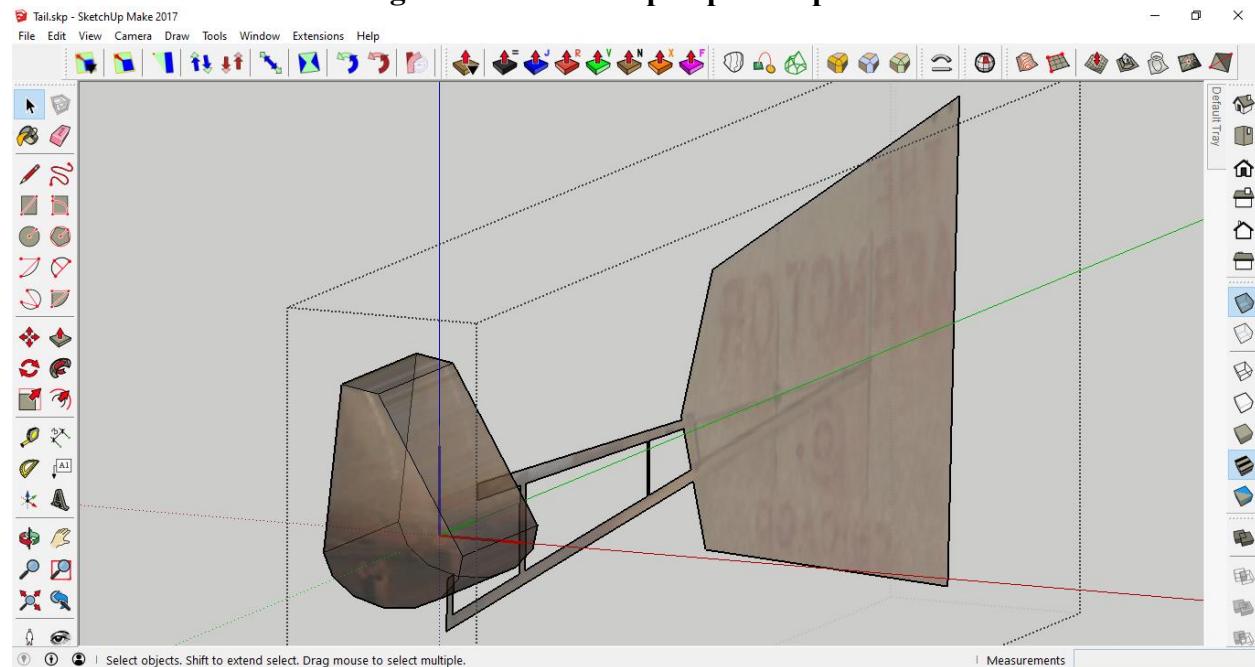
Mersul lucrării:

Pentru exemplificarea mersului lucrării de laborator nr. 4 se propune crearea unei scene dinamice care ar reprezenta o pompă de apă care își orientează elicea în dependență de direcția vîntului. Modelul pompei este divizat în trei componente: bază (Base), elice (Fan) și corp (Tail). Fiecare componentă este amplasată într-un fișier .skp aparte cu amplasarea modelului astfel încât originea sistemului de coordonate și orientarea axelor acestuia să coincidă cu originea și orientarea sistemului de coordonate global al scenei. Componentele 3D ale modelului pot fi create de la zero sau modelul poate fi importat din repozitoriu 3D

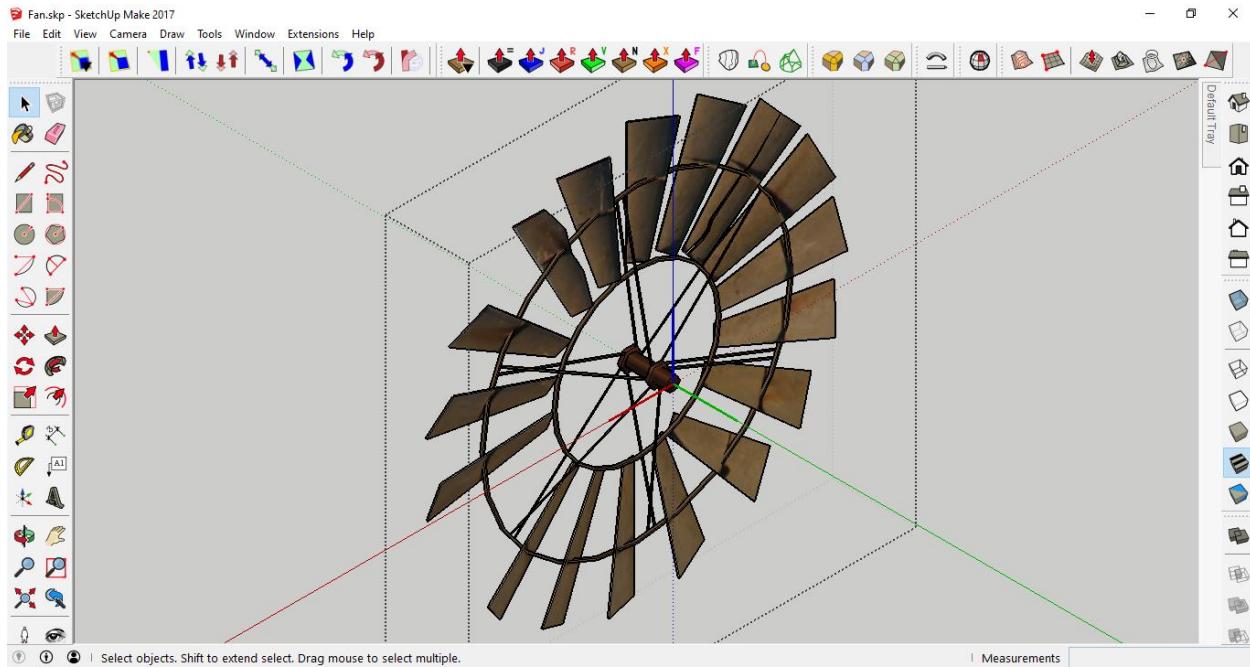
Warehouse accesibil din meniul de bază al editorului grafic 3d Google SketchUp Window->3D Warehouse, apoi editat corespunzător după necesitate.



**Fig. 4 Modelul bazei pompei de apă.**



**Fig. 5 Modelul corpului pompei de apă.**



**Fig. 6 Modelul elicei pompei de apă.**

Pentru a avea posibilitatea de a downloada sau importa modelul selectat în sketch-ul curent utilizatorul trebuie să dispună de un cont Google Account și să fie autentificat în mediul 3D Warehouse. După ce a fost downloadat și editat corespunzător fiecare componentă al modelului, acestea pot fi editate după necesitate și exportate ca fișiere .OBJ prin intermediul plug-inului *LIPID OBJ Exporter* metoda de instalare a căruia a fost prezentată în cadrul lucrării de laborator Nr 3 .

## Descrierea funcțiilor de bază din biblioteca P5.JS

### *preload()*

Apelată chiar înainte de funcția *setup()*, funcția *preload()* este utilizată pentru a gestiona încărcarea asincronă a fișierelor externe. Dacă este definită funcția de preîncărcare, funcția *setup()* va aștepta până la finalizarea oricărui apel de încărcare. Nimic în afară de apelurile de încărcare (*loadImage*, *loadJSON*, *loadFont*, *loadStrings* etc.) nu ar trebui să fie în interiorul funcției de preîncărcare. Dacă este preferată încărcarea asincronă, metodele de încărcare pot fi apelate în *setup()* sau în orice alt loc cu ajutorul unui parametru de apel invers. Nimic altceva nu ar trebui să fie realizat în interiorul funcției de *preload()* în afară de apelurile de încărcare, toate celelalte inițializări ar trebui să fie realizate în interiorul funcției *setup()*.

### Sintaxă:

*preload()*

### *loadModel()*

Încărcarea unui model 3D dintr-un fișier OBJ sau STL. Funcția *loadModel()* trebuie plasată în interiorul funcției *preload()*. Acest lucru permite modelului să se încarce complet înainte de a rula restul programului.

Una dintre limitările formatului OBJ și STL este că nu are un sens de scară încorporat. Aceasta înseamnă că modelele exportate din diferite editoare grafice 3D pot avea dimensiuni diferite. Dacă modelul nu se afișează, metoda *loadModel()* poate fi apelată cu valoarea true a parametrului de normalizare. Aceasta va redimensiona modelul la o scară adecvată pentru p5. De asemenea, pot fi făcute modificări suplimentare ale dimensiunii finale a modelului utilizând funcția *scale()*. Un alt neajuns îl reprezintă faptul că biblioteca p5.js nu asigură suportul pentru fișiere STL colorate și fișiere OBJ în care conțin definiția materialelor, texturilor și culorilor. Acestea vor fi redate fără proprietăți de materiale, texturi și culori.

### Sintaxă:

```
loadModel(path, normalize, [successCallback], [failureCallback], [fileType])  
loadModel(path, [successCallback], [failureCallback], [fileType])
```

### Parametrii:

path	String: Calea modelului de încărcat
normalize	boolean: dacă este adevărat, se scalează modelul la o dimensiune standardizată la încărcare
funcția successCallback	(p5.Geometry): funcție care trebuie apelată după încărcarea modelului. Va fi transmis obiectul model 3D. (Optional)
failureCallback	(Event): funcție apelată cu eveniment eroare de dacă modelul nu reușește să se încarce. (Optional)
fileType	String: Extensia de fișier a modelului (.stl, .obj). (Optional)

### Returnează:

obiect de tipul p5.Geometry

### *setup()*

Funcția *setup()* este apelată o singură dată când pornește programul. Este folosită pentru a defini proprietățile mediului inițial, cum ar fi dimensiunea ecranului, culoarea de fundal și pentru a încărca suporturi media, cum ar fi imagini și fonturi, pe măsură ce rulează programul. Nu poate exista decât o singură funcție *setup()* în fiecare program și nu trebuie apelată repetat după executarea sa inițială.

Notă: Variabilele declarate în interiorul funcției *setup()* nu sunt accesibile în alte funcții, inclusiv în funcția *draw()*.

### Sintaxă:

```
setup()
```

### *createCanvas()*

Creează un element de canvas în documentul HTML și setează dimensiunile acestuia în pixeli. Această metodă trebuie apelată o singură dată la începutul setării inițiale. Apelarea *createCanvas()* de mai multe ori într-un proiect va avea ca rezultat un comportament imprevizibil. Pentru crearea mai multor pânze de desen, poate fi utilizată funcția *createGraphics* (ascuns în mod implicit, dar poate fi afișat).

În modul 2D (adică când p5.Renderer nu este setat) originea (0,0) este poziționată în partea stângă sus a ecranului. În modul 3D (adică când p5.Renderer este setat la WEBGL), originea este poziționată în centrul pânzei.

Variabilele de sistem lățime și înălțime sunt setate de parametrii acestei funcții. Dacă *createCanvas()* nu este utilizat, ferestre va primi o dimensiune implicită de 100x100 pixeli.

### Sintaxă:

```
createCanvas (w, h, [renderer])
```

### Parametrii:

w	Number: lățimea pânzei
h	Number: înălțimea pânzei
render	constant: P2D, sau WebGL (optional)

### Returnează:

p5.Renderer:

### *normalMaterial()*

Materialul normal pentru geometrie este un material care nu este afectat de lumină. Nu este reflectant și este un material substituent folosit adesea pentru depanare. Suprafețele

orientate spre axa X devin roșii, cele orientate spre axa Y devin verzi și cele orientate spre axa Z devin albastre.

**Sintaxă:**

normalMaterial()

***frameRate()***

Specifică numărul de cadre care trebuie afișate în fiecare secundă. De exemplu, apelul funcției ***frameRate(30)*** va încerca să redeseneze scena de 30 de ori pe secundă. Dacă procesorul nu este suficient optimizat pentru a menține rata specificată, rata cadrelor nu va putea fi asigurată. Se recomandă setarea ratei cadrelor în interiorul funcției ***setup()***. Rata implicită a cadrelor se bazează pe rata cadrelor a afișajului (numită aici și „rată de reîmprospătare”), care este setată la 60 de cadre pe secundă pe majoritatea computerelor. O rată de cadre de 24 de cadre pe secundă (de obicei pentru filme) sau mai mare va fi suficientă pentru animații fluide. Această funcție este analogică funcției ***setFrameRate (val)***.

Apelul funcției ***frameRate()*** fără argumente returnează frecvența curentă a cadrelor. Funcția ***draw()*** trebuie să ruleze cel puțin o dată înainte ca funcția ***frameRate()*** să returneze o valoare. Această funcție este analogică funcției ***getFrameRate()***. Apelul funcției ***frameRate()*** cu argumente care nu sunt de tipul ***number*** sau care nu sunt pozitive returnează, de asemenea, frecvența curentă a cadrelor.

**Sintaxă:**

***frameRate (fps)***

***frameRate ()***

**Parametri:**

Number ***fps***: numărul de cadre care trebuie afișate în fiecare secundă

***angleMode()***

Setează modul curent de reprezentare a unității de măsură a unghiurilor în p5 în radiani sau grade. Modul implicit reprezentare a unității de măsură a unghiurilor este în radiani.

**Sintaxă:**

***angleMode (mode)***

**Parametri:**

mode constant: RADIANS, sau DEGREES

***draw()***

Apelată direct după ***setup()***, funcția ***draw()*** execută continuu liniile de cod conținute în blocul său până când programul este oprit sau se apelează ***noLoop()***. Dacă funcția ***noLoop()*** este apelată în interiorul funcției ***setup()***, funcția ***draw()*** va fi executată încă o dată înainte de oprire. Funcția ***draw()*** este apelată automat și nu trebuie apelată în mod explicit. Apelul funcției ***draw()*** ar trebui să fie întotdeauna controlat cu ***noLoop()***, ***redraw()*** și ***loop()***. După ce ***noLoop()*** oprește executarea codului din ***draw()***, ***redraw()*** face ca codul din ***draw()*** să se execute o singură dată, iar apelul funcției ***loop()*** va relua executarea ciclică ca codul din funcția ***draw()***. Numărul de execuții al funcției ***draw()*** în fiecare secundă poate fi controlat cu ajutorul funcției ***frameRate()***. Poate exista o singură funcție ***draw()*** pentru fiecare scenă în fiecare proiect. Funcția ***draw()*** trebuie să existe dacă pentru ca codul să ruleze continuu sau să proceseze evenimente de tipul ***mousePressed()***.

Trebuie specificat faptul că toate transformările realizate asupra sistemului de coordonate al proiectului vor fi resetate la începutul fiecărui apel al funcției ***draw()***. Dacă se efectuează transformări în cadrul funcției ***draw()*** (ex: scalare, rotire, translare), efectele acestora vor fi anulate la începutul funcției ***draw()***, astfel transformările nu se vor acumula în timp. Pe de altă parte, stilul aplicat (ex: umplere, linie, etc.) va rămâne în vigoare.

### Sintaxă:

draw()

### ***orbitControl()***

Permite mișcarea în jurul unei scene 3D folosind un mouse sau un trackpad. Făcând clic stânga și mișcând mouse-ul se va roti poziția camerei în jurul centrului scenei, făcând clic dreapta și mișcând mouse-ul se va deplasa poziția camerei fără rotație, iar folosind roata mouse-ului (scrolling) se va mișca camera mai aproape sau mai departe de centrul scenei. Această funcție poate fi apelată cu parametri care specifică sensibilitatea la mișcarea mouse-ului de-a lungul axelor X și Y. Apelarea acestei funcții fără parametri este echivalentă cu apelul funcției ***orbitControl(1, 1)***. Pentru a inversa direcția de mișcare în ambele axe, trebuie introdusă o valoare negativă pentru sensibilitate.

### Sintaxă:

orbitControl([sensitivityX], [sensitivityY], [sensitivityZ])

### Parametrii:

sensitivityX (Optional)	Number: sensibilitate la mișcarea mouse-ului de-a lungul axei X
sensitivityY (optional)	Number: sensibilitate la mișcarea mouse-ului de-a lungul axei Y
sensitivityZ (optional)	Number: sensibilitate la mișcarea de derulare de-a lungul axei Z

### ***background()***

Funcția ***background()*** setează culoarea utilizată pentru fundalul scenei în p5.js. Fundalul implicit este transparent. Această funcție este de obicei utilizată în interiorul funcției ***draw()*** pentru a șterge fereastra de afișare a fiecărui cadru, dar poate fi utilizată și în interiorul funcției ***setup()*** pentru a seta fundalul primului cadru al animației sau dacă fundalul trebuie setat o singură dată.

Culoarea este fie specificată în unul din formatele RGB, HSB sau HSL, în funcție de modul color actual setat. Formatul de culoare implicit este RGB, cu fiecare valoare în intervalul de la 0 la 255. În mod implicit, intervalul *alfa* variază de la 0 la 255.

Dacă este furnizat un singur argument, sunt acceptate formatele de culori RGB, RGBA și Hex CSS și toate formatele de culoare denumite. În acest caz, nu este acceptată valoarea pentru parametrul *alfa* ca al doilea argument, trebuie utilizat formularul RGBA.

Un obiect p5.Color poate fi, de asemenea, utilizat pentru a seta culoarea de fundal.

Puteți utiliza și o imagine în p5.js pentru a seta imaginea de fundal.

### Sintaxă:

background(color)  
background(colorstring, [a])  
background(gray, [a])  
background(v1, v2, v3, [a])  
background(values)  
background(image, [a])

### Parametri:

color	p5.Color: orice valoare creată de funcția color()
colorstring	String: sir de culoare, formatele posibile includ: întreg rgb() sau rgba(), procent rgb() sau rgba(), hex de 3 cifre, hex de 6 cifre
a	Number: opacitatea fundalului în raport cu gama de culori curentă (implicit este 0-255) (Optional)
gray	Number: specifică o valoare între alb și negru
v1	Number: roșu sau nuanță (în funcție de formatul de culoare curent)

v2 culoare curent)	Number: verde sau valoare de saturăție (în funcție de formatul de culoare curent)
v3	Number: albastru sau valoarea luminozității (în funcție de formatul de culoare curent)
values	Number[]: o matrice care conține componentele roșu, verde, albastru și alfa ale culorii
image	p5.Image: imagine creată cu ajutorul funcției <b>loadImage()</b> sau <b>createImage()</b> , pentru a seta ca fundal (imaginile trebuie să fie de aceeași dimensiune ca scena)

### ***noStroke()***

Dezactivează desenarea liniei de contur. Dacă sunt apelate atât funcția **noStroke()**, cât și **noFill()**, nimic nu va fi afișat pe ecran.

#### **Sintaxă:**

`noStroke ()`

### ***push() și pop()***

Funcția **push()** salvează setările și transformările curente ale stilului de desen, în timp ce **pop()** restabilește aceste setări. Trebuie menționat faptul că aceste funcții sunt utilizate întotdeauna împreună. Aceste funcții permit modificarea stilului și transformărilor pentru a reutiliza ulterior la setările anterioare. Când o nouă stare este pornită cu **push()**, aceasta se bazează pe stilul curent și transformă informațiile. Funcțiile **push()** și **pop()** pot fi încorporate pentru a oferi mai mult control.

Funcția **push()** stochează informații legate de transformările curente și setările de stil controlate de următoarele funcții: **fill()**, **noFill()**, **noStroke()**, **stroke()**, **tint()**, **noTint()**, **strokeWeight()**, **strokeCap()**, **strokeJoin()**, **imageMode()**, **rectMode()**, **ellipseMode()**, **colorMode()**, **textAlign()**, **textFont()**, **textSize()**, **textLeading()**, **applyMatrix()**, **resetMatrix()**, **rotate()**, **scale()**, **shearX()**, **shearY()**, **translate()**, **noiseSeed()**.

În modul **WEBGL** sunt stocate setări de stil suplimentare. Acestea sunt controlate de următoarele funcții: **setCamera()**, **ambientLight()**, **directionalLight()**, **pointLight()**, **texture()**, **specularMaterial()**, **shininess()**, **normalMaterial()** și **shader()**.

#### **Sintaxă:**

`push()  
pop()`

### ***scale()***

Mărește sau scade dimensiunea unui obiect prin extinderea sau contractarea vârfurilor. Obiectele se scalează întotdeauna în raport cu originea lor relativă a sistemului de coordonate. Valorile scalei sunt specificate ca procente zecimale. De exemplu, apelul funcției **scale(2.0)** mărește dimensiunile unui obiect cu 200%.

Transformările se aplică la tot ceea ce se întâmplă după și apelurile ulterioare ale funcție amplifică efectul acestora. De exemplu, apelul funcției **scale(2.0)** și apoi **scale(1.5)** este echivalent cu **scale(3.0)**. Dacă funcția **scale()** este apelată în interiorul funcției **draw()**, transformarea este resetată la fiecare apel al acestei funcții.

Utilizarea acestei funcții cu parametrul **z** este disponibilă numai în modul **WEBGL**. Această funcție poate fi controlată suplimentar cu ajutorul funcției **push()** și **pop()**.

#### **Sintaxă:**

`scale(s, [y], [z])  
scale(scales)`

#### **Parametri:**

s	Number   p5.Vector   Number[]]: procent pentru a scala obiectul sau procent pentru a scala obiectul pe axa x dacă sunt date mai multe argumente
y	Number: procentul de scalare a obiectul pe axa y (optional)
z (optional)	Number: procentul de scalare a obiectul pe axa z (numai în regim webgl)
scales	p5.Vector   Number[]]: procentul de scalare a obiectului

### ***translate()***

Specifică valoarea pentru deplasarea obiectelor din scenă. Parametrul **x** specifică translarea stânga/dreapta, parametrul **y** specifică translarea sus/jos.

Transformările sunt cumulative și se aplică la tot ceea ce se întâmplă după și apelurile repetitive către funcția dată acumulează efectul. De exemplu, apelarea **translate(50, 0)** și apoi a **translate(20, 0)** este echivalent cu apelul **translate(70, 0)**. Dacă **translate()** este apelat în interiorul funcției **draw()**, transformarea este resetată în momentul apelului repetat al acestei funcții. Această funcție poate fi controlată cu ajutorul funcțiilor **push()** și **pop()**.

#### **Sintaxă:**

**translate(x, y, [z])**  
**translate(vector)**

#### **Parametri:**

x	Number: translează la stânga/dreapta
y	Number: translează în sus/jos
z	Number: translează înainte/înapoi (numai webgl) (optional)
vector	p5.Vector: vectorul cu care se translează

### ***rotate()***

Rotește un obiect cu cantitatea specificată de parametrul unghiului. Această funcție reprezintă **angleMode**, astfel încât unghiiurile pot fi introduse fie în radiani (RADIAN), fie în grade (DEGREES).

Obiectele sunt întotdeauna rotite în jurul poziției lor relative la origine, iar numerele pozitive rotesc obiectele în sensul acelor de ceasornic. Transformările se aplică la tot ceea ce se întâmplă după și apelurile repetitive ale acestei funcții cumulează efectul acestora. De exemplu, apelarea **rotate(HALF\_PI)** și apoi **rotate(HALF\_PI)** este echivalent cu **rotate(PI)**. Toate transformările sunt resetate la apelul repetat al funcției **draw()**.

Tehnic, funcția **rotate()** realizează înmulțirea matricei de transformare curente cu o matrice de rotație. Această funcție poate fi controlată cu ajutorul funcțiilor **push()** și **pop()**.

#### **Sintaxă:**

**rotate(angle, [axis])**

#### **Parametri:**

angle	Number: unghiul de rotație, specificat în radiani sau grade, în funcție de modul curent de reprezentarea a unghiiurilor
axis	p5.Vector   Număr []: (în regim 3d) axa de rotit (Optional)

### ***rotateX(), rotateY(), rotateZ()***

Rotește un obiect în jurul axei **X** cu valoarea unghiului specificată în parametrul **angle**. Unghiiurile pot fi introduse fie în radiani (RADIAN), fie în grade(DEGREES).

Obiectele sunt întotdeauna rotite în jurul poziției lor relative la origine, iar numerele pozitive rotesc obiectele în sensul acelor de ceasornic. Toate transformările sunt resetate înaintea unui nou apel al funcției **draw()**.

#### **Sintaxă:**

**rotateX(angle)**  
**rotateY(angle)**  
**rotateZ(angle)**

**Parametri:**

angle Number: unghiul de rotație, specificat în radiani sau grade, în funcție de modul curent de reprezentare a unghiurilor

***model()***

Redă un model 3D în scenă.

**Sintaxă:**

model(model)

**Parametri:**

model p5.Geometrie: Modelul 3D încărcat care trebuie redat

**Listingul programului:**

```
let fr = 30;
let angle = 0;
function preload()
{
  base = loadModel('Base.obj');
  fan = loadModel('Fan.obj');
  tail = loadModel('Tail.obj');
}
function setup()
{
  createCanvas(400, 400, WEBGL);
  normalMaterial();
  frameRate(fr);
  angleMode(DEGREES);
}
function draw()
{
  angle = 45*sin(millis()/50);
  orbitControl();
  background(50);
  noStroke();
  push();
  scale(0.025);
  translate(0, 0, -4000);
  model(base);
  pop();
  push();
  scale(0.025);
  rotateZ(angle);
  translate(0, -200, 4650);
  rotateY(millis()/10);
  model(fan);
  pop();
  push();
  scale(0.025);
  translate(0, 0, 4650);
  rotateZ(angle);
  model(tail);
  pop();
```

}

### Descrierea programului:

Variabila **fr** păstrează numărul de cadre redate pe secundă, numărul de apeluri a funcției **draw()** pentru redarea scenei.

Variabila **angle** este destinată pentru păstrarea unghiului de abatere (orientare) a corpului pompei de vînt și a elicei.

În funcția **preload()** destinată pentru încărcarea modelelor obiectelor 3D ale bazei (**Base.obj**), elicei (**Fan.obj**) și corpului (**Tail.obj**) modelului pompei de apă cu acțiune eoliană stocate în fișiere .OBJ, Fig. ?.

Modul de adăugare în proiect a fișierelor .obj care conțin geometria obiectelor 3D este descris în cadrul lucrării de laborator nr. 3.

În funcția **setup()** care este apelată o singură dată la lansarea programului, sînt realizate toate configurațiile necesare pentru a lucra în regim 3D cum ar fi apelul funcției **createCanvas(400, 400, WEBGL)** care creează o scenă 3D redată cu ajutorul unui canvas cu dimensiunile 400 pe 400 de pixeli.

Este apelată funcția **normalMaterial()** cu ajutorul coreia este setat un material normal pentru geometrie care este un material care nu este afectat de lumină, nu este reflectant și iar suprafețele orientate spre axa **X** devin roșii, cele orientate spre axa **Y** devin verzi și cele orientate spre axa **Z** devin albastre.

Cu ajutorul funcției **frameRate(fr)** este setată rata de redare a cadrelor egală cu 30 de cadre pe secundă.

Cu ajutorul funcției **angleMode(DEGREES)** este setat modul de reprezentare a gradelor în grade.

Ulterior este apelată repetat cu frecvență de 30 de ori pe secundă funcția **draw()** care desenează repetat scena. Variabila **angle** stochează valoarea în grade a unghiului de orientare a pompei. Această valoare variază în timp în dependență de timpul care a trecut de la începutul lansării programului reprezentat în milisecunde. Valoarea unghiului variază în timp în intervalul de la -45 la +45 de grade. Această variație este realizată cu ajutorul instrucțiunii **angle = 45\*sin(millis()/50)** care și asigură animația în cadrul scenei.

Este utilizată funcția **orbitControl()** care asigură controlul și poziționarea camerei din cadrul scenei. Cu ajutorul funcției **background(50)** este setată culoarea sură a fundalului. Cu ajutorul funcției **noStroke()** este setat modul grafic de redare a muchiilor astfel încît acestea să nu fie vizibile. Pentru aplicarea individuală a transformărilor pentru fiecare obiect în parte, acestea se încadrează fiecare în interiorul unui bloc **push()**, **pop()**. În interiorul acestei construcții sunt realizate toate transformările necesare pentru fiecare obiect grafic. Pentru încararea obiectelor în scenă toate obiectele sunt scalate cu ajutorul funcției **scale(0.025)** care micșorează dimensiunile obiectelor cu coeficientul respectiv. Modelul bazei pompei de apă este translat pe axa **Z** în jos cu 4000 de unități pentru a fi centrat în vizorul camerei. Modelul bazei este redat în scenă cu ajutorul apelului funcției **model(base)**.

Pentru reprezentarea elicei, asupra acesteia sunt realizate o serie de transformări cum ar fi scalarea, rotația modelului deja scalat în jurul axei **Z** cu valoarea unghiului păstrată în variabila **angle**, apoi elicea este translată cu ajutorul funcției **translate(0, -200, 4650)** cu 200 de unități înainte pe axa **Y** și cu 4650 de unități în sus pe axa **Z**. Ulterior cu ajutorul funcției **rotateY(millis()/10)** elicea este rotită în jurul axei **Y** al poziției relative la origine și simulează influența vîntului asupra elicei.

Pentru reprezentarea corpului pompei dimensiunile acestuia sunt scalate cu același coeficient și apoi translată în sus pe axa **Z** cu ajutorul funcției **translate(0, 0, 4650)**, apoi rotită în jurul axei **Z** cu ajutorul funcției **rotateZ(angle)**.

Toate modelele sunt redate în scenă cu ajutorul funcției **model()** apelat fiecare în blocul său **push()**, **pop()**.

The screenshot shows the p5.js editor interface. On the left, the 'Sketch Files' sidebar lists files: Base.obj, Fan.obj, Tail.obj, index.html, sketch.js (selected), and style.css. The main area displays the code for 'sketch.js':

```

function draw() {
  angle = 45 * sin(millis()) / 50;
  orbitControl();
  background(50);
  noStroke();
  push();
  scale(0.025);
  translate(0, 0, -4000);
  model(base);
  pop();

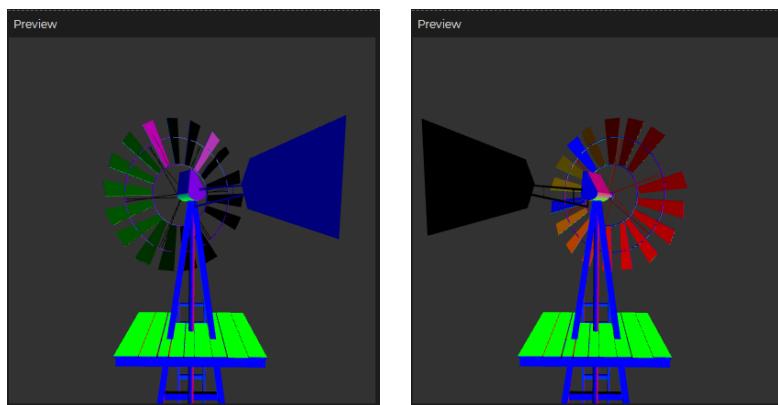
  push();
  scale(0.025);
  rotateZ(angle);
  translate(0, -200, 4650);
  rotate(millis() / 10);
  model(fan);
  pop();

  push();
  scale(0.025);
  translate(0, 0, 4650);
  rotateZ(angle);
  model(tail);
  pop();
}

```

The preview window on the right shows a 3D wireframe model of a windmill with a blue tower, green base, and multi-colored blades.

**Fig. 7 Modelul pompei de apă în editorul p5.js.**



**Fig. 8 Rezultatul execuției programului în editorul p5.js.**

## Lucrarea de laborator nr. 4

### Tema: Transformări 3D

Scopul lucrării: Obținerea cunoștințelor practice în sinteza scenelor grafice 3D dinamice, utilizând funcțiile standard de translație, rotație și scalare din biblioteca p5.js.

#### Sarcina lucrării:

1. Elaborați un program pentru sinteza unei scene 3D dinamice utilizând funcțiile standard de translație, rotație și scalare din biblioteca p5.js.
2. Elaborați un program care crează o scenă 3D dinamică conform variantei indicate în tabelul 3.1. Pentru crearea scenei pot fi utilizate obiecte grafice 3D existente în repozitoriul 3D.

