

UNIVERSITATEA TEHNICĂ A MOLDOVEI

**Rotaru Lilia, Oșovschi Mariana,
Cărbune Viorel, Ursu Adriana**

**GRAFICA
PE CALCULATOR**

**ÎNDRUMAR METODIC
PENTRU LUCRĂRILE DE LABORATOR**



**Chișinău
2025**

UNIVERSITATEA TEHNICĂ A MOLDOVEI
FACULTATEA CALCULATOARE, INFORMATICĂ
ȘI MICROELECTRONICĂ
DEPARTAMENTUL INFORMATICĂ ȘI INGINERIA
SISTEMELOR

Rotaru Lilia, Oșovschi Mariana,
Cărbune Viorel, Ursu Adriana

GRAFICA
PE CALCULATOR

ÎNDRUMAR METODIC
PENTRU LUCRĂRILE DE LABORATOR

Chișinău
Editura „Tehnica-UTM”
2025

CZU

Îndrumarul metodic pentru desfășurarea lucrărilor de laborator la disciplina *Grafica pe calculator* este destinat studenților programelor de studii 0612.3 *Știința datelor*, 0714.7 *Robotică și mecatronică*, 0613.5 *Informatică aplicată*, 0714.6 *Automatică și informatică*, 0612.1 *Calculatoare și rețele*, 0714.4 *Electronica aplicată*, 0714.9 *Inginerie biomedicală*, 0612.2 *Managementul informației*, 0714.5 *Microelectronică și nanotehnologii*, 0613.1 *Tehnologia informației*. Îndrumarul are scopul de a ajuta studenții să înțeleagă principiile de bază ale creării modelelor grafice 2D, 3D, realitatea augmentată și elaborarea modelelor pentru imprimarea 3D. De asemenea, include informații teoretice concise, stabilește ordinea în care trebuie efectuate lucrările de laborator și cerințele privind rapoartele de laborator. Acestea sunt explicate într-un limbaj accesibil, ținând cont de faptul că se acordă o atenție deosebită muncii individuale a studenților în procesul de învățare.

Autori: asist. univ. Lilia ROTARU
 asist. univ. Mariana OȘOVȘCHI
 lect.univ., dr. Viorel CĂRBUNE

Redactor responsabil: conf.univ., dr. Viorica SUDACEVȘCHI

Recenzenți: conf.univ., dr. Ion BODNARIUC
 conf.univ., dr. Irina COJUHARI

INTRODUCERE

Grafica pe calculator este un domeniu care se ocupă cu crearea, manipularea, prelucrarea și afișarea imaginilor prin intermediul calculatoarelor. Acest domeniu este în continuă dezvoltare și implică generarea și modificarea graficii digitale bidimensionale și tridimensionale. Grafica utilizează diverse tehnici și algoritmi pentru generarea imaginilor folosind calculatorul. În prezent, grafica pe calculator are o aplicabilitate extinsă, de la jocuri video și animație până la simulări științifice, proiectare asistată de calculator (CAD), vizualizare de date și efecte speciale în filme. Îndrumarul este bazat pe biblioteca grafică P5.js la baza căreia se află Java Script.

Domeniul cuprinde noțiuni precum:

- *Primitive grafice*: formele de bază, cum ar fi puncte, linii, cercuri, poligoane utilizate pentru a construi imagini mai complexe.
- *Transformări grafice*: manipulări geometrice care permit translația, rotația, scalarea și distorsionarea obiectelor grafice.
- *Modelare 3D*: crearea obiectelor tridimensionale prin intermediul unor tehnici avansate precum extrudarea, sculptarea și texturarea.
- *Iluminarea și umbrirea*: simularea efectelor de lumină, umbră și reflecție pentru a crea imagini realiste.

Un domeniul nou al graficii care se dezvoltă rapid în ultimul timp este realitatea augmentată (RA) – tehnologie care îmbină elementele virtuale generate pe calculator cu lumea reală, oferind o experiență interactivă și îmbunătățită. RA utilizează camere și senzori pentru a suprapune obiectele 2D sau 3D pe fluxul de imagine al camerei unui dispozitiv, creând iluzia că aceste obiecte sunt integrate în mediul real. Tehnologia este utilizată într-o varietate de aplicații.

Lucrarea include considerații teoretice și sarcini pentru lucrările de laborator la disciplina *Grafica pe calculator* și este destinată studenților FCIM, ciclul I licență, care urmează programele de studii: 0612.3 *Știința datelor*, 0714.7 *Robotică și mecatronică*, 0613.5 *Informatică aplicată*, 0714.6 *Automatică și informatică*,

0612.1 Calculatoare și rețele, 0714.4 Electronica aplicată, 0714.9 Inginerie biomedicală, 0612.2 Managementul informației, 0714.5 Microelectronică și nanotehnologii, 0613.1 Tehnologia informației și alte programe de studii care implică grafica pe calculator.

Lucrarea conține 7 capitole, fiecare capitol finalizează cu sarcini și instrucțiuni privind executarea lucrărilor de laborator, precum și criteriile de evaluare a acestora.

Lucrările de laborator sunt structurate astfel, încât să permită studenților să se familiarizeze treptat cu diferitele concepte și tehnici utilizate în grafică pe calculator, începând cu elementele de bază ale graficii 2D, continuând cu reprezentările 3D, modalitățile de realizare a scenelor dinamice și culminând cu realitatea augmentată și realizarea modelelor pentru imprimante 3D. Fiecare lucrare include sarcini practice și descrieri detaliate ale metodelor și funcțiilor necesare, asigurând astfel o înțelegere profundă a fiecărui concept, precum și criteriile de evaluare a sarcinilor realizate.

La finalizarea lucrărilor de laborator, studenții vor dezvolta competențe teoretice și practice esențiale în domeniul graficii pe calculator aplicabile în diverse domenii tehnologice și profesionale.

Cerințe de securitate la realizarea experimentelor sau îndeplinirea lucrărilor

Cerințele de securitate se bazează pe Legea nr. 186 din 2008, Hotărârea de Guvern nr. 95 din 2009 și Hotărârea de Guvern nr. 252 din 2010, și anume:

Norme de securitate înainte de începutul lucrărilor

1. Este interzisă efectuarea lucrărilor de laborator în absența conducătorului de lucrări (lector sau laborant).
2. Înainte de începutul lucrării, studenții vor studia lucrarea respectivă. Studenții nepregătiți pentru lucrare nu vor fi admiși la efectuarea lucrărilor.
3. Cuplarea montajului experimental al lucrării de laborator la rețeaua electrică se face numai cu permisiunea conducătorului de lucrări.

Normele de securitate în procesul lucrului

1. Este interzisă lăsarea aparatului sub tensiune, schimbarea locului de lucru fără permisiunea conducătorului de lucrări.
2. Se interzice schimbarea siguranței de alimentare când aparatul este cuplat în rețea.
3. Se interzice atingerea părților neizolate ale circuitului electric.
4. Se interzice a supune radiației părțile corpului, îndeosebi ochii.
5. Se interzice a pune mâna pe suprafața sursei radioactive.
6. Sursa radioactivă este primită de la laborant la începutul lucrării și este strict necesar a o întoarce după terminarea lucrării.

Normele de securitate la terminarea lucrărilor

1. La sfârșitul lucrărilor se decuplează montajul experimental de la tensiunea electrică, se blochează cronometrele electrice digitale.
2. Se deconectează de la rețea firul de legătură; se scoate din priză cordonul cablului electric cu contrafișă.

3. Conducătorul lucrărilor și laborantul trebuie să verifice dacă au fost deconectate aparatele și dacă a fost stinsă lumina. În afară de aceasta, se vor asigura că nu există miros de materiale arzânde, scânteii și alte semnale de aprindere.

Acțiuni întreprinse în caz de accidente și alte situații critice

1. În cazul când o persoană este sub tensiune electrică, trebuie să se deconecteze imediat montajul experimental și să se anunțe conducătorul de lucrări. Dacă accidentatul nu și-a pierdut conștiința, mai întâi i se asigură liniște, apoi este trimis la medic sau este chemată asistența medicală de urgență prin telefonul 112.

2. Dacă accidentatul și-a pierdut conștiința și nu respiră, atunci după scoaterea de sub tensiune imediat i se face respirație artificială, măsuri de oprire a scurgerilor de sânge. Concomitent este chemată asistența medicală de urgență prin numărul de telefon 112.

3. În caz de răniri, vânătăi și alte leziuni trebuie folosită trusa medicală; se va acționa conform indicațiilor de acordare a primului ajutor.

4. La apariția mirosului, scânteilor, a fumului sau altor fenomene anormale montajul se decuplează imediat.

5. În caz de incendiu, trebuie folosit mai întâi extingtorul (stingătorul) cu bioxid de carbon, iar în locurile fără instalații electrice se va folosi extingtorul cu spumă, nisip, apă sau pătura de azbest.

Modalitatea de evaluare a cunoștințelor și rezultatelor activității practice

Pentru fiecare lucrare de laborator se va elabora un raport în baza modelului reprezentat în anexa 1.

Modalitatea de evaluare a lucrărilor de laborator constă în următoarele:

Studentul va prezenta codul programului la calculator, profesorul va evalua studentul în baza:

✓ răspunsurilor la întrebările de autoevaluare care sunt expuse după fiecare lucrare de laborator;

✓ profesorul va evalua codul în dependență de respectarea condițiilor din sarcinile propuse spre realizare, în dependență de funcționalitatea și optimizarea programului precum și criteriilor de evaluare enumerate în fiecare lucrare de laborator.

Studentul trebuie să respecte termenul-limită de două săptămâni pentru prezentarea lucrării de laborator. Pentru fiecare săptămână de întârziere profesorul este în drept să micșoreze punctajul studentului pentru lucrarea respectivă.

<i>Criterii de evaluare</i>	<i>Descriptori</i>				<i>Ponderea criteriului de evaluare în nota finală acordată candidatului</i>
	<i>Nivel maxim (nota 9.00-10)</i>	<i>Nivel mediu (nota 7.00-8.99)</i>	<i>Nivel minim (nota 5.00-6.99)</i>	<i>Nivel insuficient (nota <5.00)</i>	
Compilarea și rularea programului	Compilarea fără erori, afișarea corectă a rezultatului rulării programului și respectarea tuturor cerințelor lucrării	Compilarea fără erori, afișarea corectă a rezultatului rulării programului și respectarea majorității cerințelor lucrării	Compilarea fără erori, afișarea corectă a rezultatului rulării programului și respectarea parțială a cerințelor lucrării	Compilarea cu erori a programului	0.2
Conținutul raportului	Respectarea conținutului și standardelor de formatare a raportului	Respectarea parțială a conținutului și standardelor de formatare a raportului	Nerespectarea parțială a conținutului și standardelor de formatare a raportului	Nerespectarea totală a conținutului și standardelor de formatare a raportului	0.2
Susținerea raportului	Expunerea cursivă, ordonată logic și exhaustivă a raportului.	Expunerea cursivă și încrezătoare a raportului.	Face pauze lungi în expunerea raportului	Nu este în stare să expună raportul	0.3
Răspunsul la întrebări la tema lucrării de laborator	Răspunde prompt și corect la toate întrebările formulate	Răspunde corect la mai mult de 50% din întrebările formulate	Răspunde corect la 50% din întrebările formulate	Nu poate răspunde la întrebările formulate	0.3

I. NOȚIUNI GENERALE PRIVIND BIBLIOTECILE GRAFICE P5.JS

P5.js este o bibliotecă grafică 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. Acesta înseamnă integrarea simplă a codurilor în pagini web prin adăugarea acestuia într-un document HTML.

P5.js este gratis, open-sources și independent de platformă, deci, aplicațiile pot fi rulate pe orice sistem de operare, de asemenea p5.js are o familie mare de limbaje și medii programare înrudite, cum ar fi C++, JavaScript, Processing, Python etc.

Crearea unui program simplu în P5.js

Înainte de a începe crearea programelor proprii trebuie să se cunoască 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 reprezentate în figura 1.1

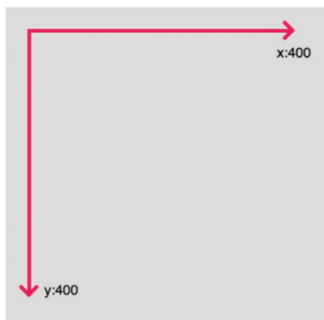


Figura 1.1 Sistemul de coordonate în mediul p5.js

Pentru crearea unui program simplu în p5.js vom utiliza următorul șablon:

```
function setup(){
```

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

Funcția setup() – este apelată o singură dată la începutul programului, se folosește pentru setarea proprietăților inițiale ale mediului de lucru, cum ar fi dimensiunea și culoarea ecranului, încărcarea fișierelor multimedia la lansarea programul, imaginile și fonturile. Poate exista o singură funcție **setup()** per program și nu trebuie apelată după execuția inițială.

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

Funcția createCanvas() – creează un element canvas (pânză) și setează dimensiunea acestuia în pixeli. Această metodă este 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**.

Sintaxa:

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

unde:

w – număr, lățimea pânzei;

h – număr, înălțimea pânzei.

Constanta renderului: P2D - dacă originea sistemului de coordonate se află în colțul stâng de sus al ecranului sau WEBGL – dacă originea sistemului de coordonate se află în centrul pânzei este specific pentru grafica 3D (opțională).

Dacă **createCanvas()** nu este utilizat în programul pânzei, i se atribuie valoarea implicită 100x100.

Funcția draw() se apelează imediat după **setup()**; execută în continuu rândurile de cod care sunt incluse în corpul său până la sfârșitul programului sau până la apelarea **noLoop()**.

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

Sintaxa funcției:

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

Funcția background() – setează culoarea utilizată ca fundalul canvas-ului. Implicit fundalul este transparent. Această funcție 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 în primul cadru al animației sau atunci când 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:

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

unde:

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) (opțional);
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 saturației (în dependență de gama curentă de culori);
v3 – (număr) specifică valoarea albastră sau valoarea luminozității (în dependență de gama curentă de culori);
values – un masiv care conține componentele roșie, verde, albastră și alfa a culorilor;
image – imaginea creată utilizând **loadImage()** sau **createImage()** pentru a fi setată ca fundal.

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 1.2 este arătată corespunderea punctelor figurilor geometrice și parametrilor care trebuie indicate ca argumentele funcțiilor în codul programului.

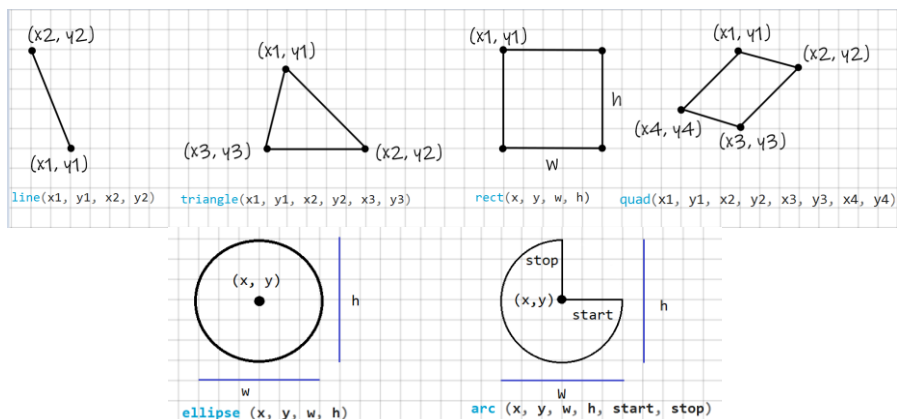


Figura 1.2. Relația dintre punctele geometrice și parametrii funcțiilor P5.js

Funcția arc(): desenează un arc pe ecran. Dacă este apelată doar cu parametrii 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). Modul de desenare poate fi modificat folosind funcția `ellipseMode()`. Diferența dintre aceste regimuri de desenare este arătată în figura 1.4.

Arcul întotdeauna este desenat în sensul acelor de ceasornic. Punctele de început ($start$) și sfârșit ($stop$) pot folosi constantele `p5.js` conform valorilor indicate în circumferința reprezentată în figura 1.3. Dacă punctele $start$ și $stop$ cad în același loc, se va desena o elipsă (cerc) completă. Axa Y crește în jos, astfel încât unghiurile sunt măsurate în sensul acelor de ceasornic din direcția X pozitivă.

Sintaxa:
arc (x, y, w, h, start, stop, [mode], [detail]);
unde:
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 al arcului;
stop (număr întreg) – unghiul de sfârșit al arcului;
mode (constantă) – parametrul care determină modul în care este desenat arcul: COORD, PIE sau OPEN (opțional);
detail (număr întreg) – parametrul opțional numai pentru modul WebGL.

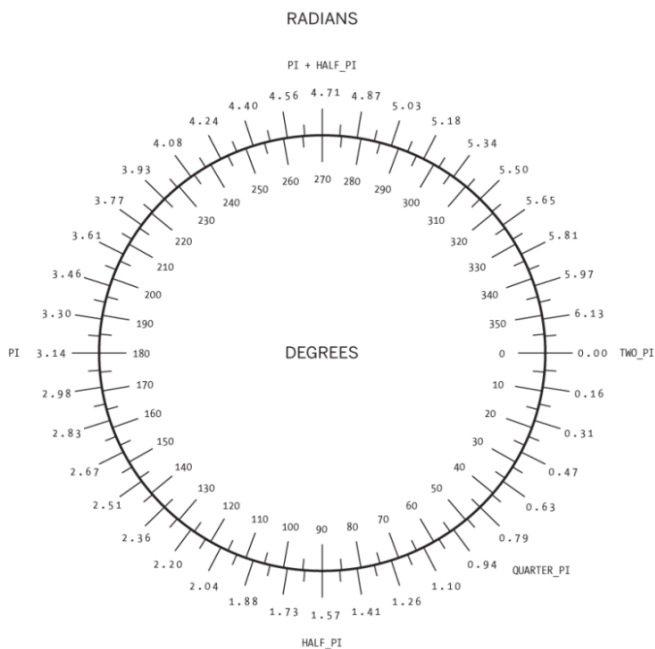


Figura 1.3. Relația dintre constante, radiani și grade în p5.js [10]

Exemplu:

```
function draw() {  
  background(220);  
  arc(50, 55, 50, 50, 0, HALF_PI); //1  
  push();  
  noFill();  
  arc(50, 55, 60, 60, HALF_PI, PI); //2  
  arc(50, 55, 70, 70, PI, PI + QUARTER_PI); //3  
  arc(50, 55, 80, 80, PI + QUARTER_PI, TWO_PI); //4  
  pop();  
  arc(50, 150, 80, 80, 0, PI + QUARTER_PI); //5  
  arc(220, 150, 80, 80, 0, PI + QUARTER_PI, PIE); //6  
  arc(50, 250, 80, 80, 0, PI + QUARTER_PI, CHORD); //7  
  arc(220, 250, 80, 80, 0, PI + QUARTER_PI, OPEN); //8  
}
```

Rezultatul rulării programului este reprezentat în figura 1.4.

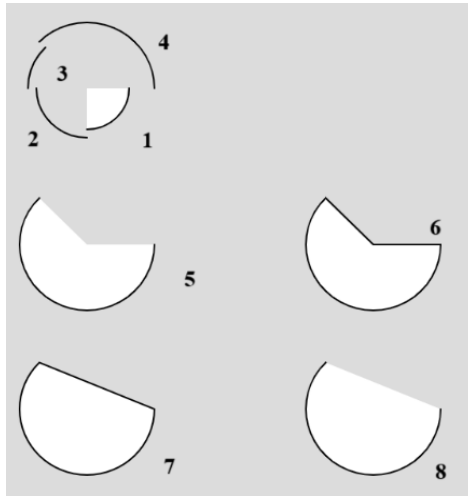


Figura 1.4. Exemple de desenare a arcurilor

Funcția ellipse(): desenează o elipsă (oval) pe ecran. Elipsa cu lățimea și înălțimea egale este un cerc. În mod implicit, primii doi parametri specifică coordonatele centrului figurii, iar al treilea și al patrulea parametru 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.

Sintaxa:

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

unde:

x (număr întreg) – coordonata x a centrului figurii;

y (număr întreg) – coordonata y a centrului figurii;

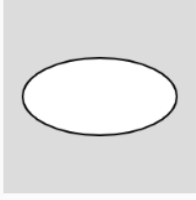
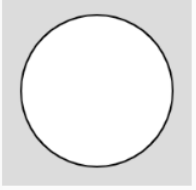
w (număr întreg) – lățimea elipsei;

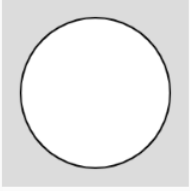
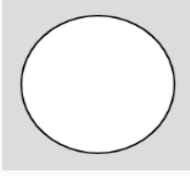
h (număr întreg) – înălțimea elipsei;

detail (număr întreg) – parametrul opțional numai pentru modul WebGL.

În tabelul 1.1 sunt aduse exemple de utilizare a funcțiilor ellipse și circle.

Tabelul 1.1. Exemple de utilizare a funcțiilor ellipse și circle și rezultatul rulării codurilor

<i>Programul</i>	<i>Rezultatul</i>
<pre>function setup() { createCanvas(100, 100); } function draw() { background(220); ellipse(50, 50, 80, 40); }</pre>	
<pre>function setup() { createCanvas(100, 100); } function draw() { background(220); ellipse(50, 50, 80); }</pre>	

<i>Programul</i>	<i>Rezultatul</i>
<pre>function setup() { createCanvas(100, 100); } function draw() { background(220); ellipse(50, 50, 80, 80);}</pre>	
<pre>function setup() { createCanvas(100, 100); } function draw() { background(220); circle (50, 50, 80); }</pre>	

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. Primii doi parametri stabilesc coordonatele centrului cercului, al treilea - diametrul cercului.

Sintaxa:

circle(x, y, d);

unde:

x (număr întreg) – coordonata x a centrului figurii;

y (număr întreg) – coordonata y a centrului figurii;

d (număr întreg) – diametrul cercului.

Funcția line(): trasează o linie dreaptă între două puncte pe ecran. Pentru a desena linii de culori, 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()**.

Sintaxa:

line(x1, y1, x2, y2);

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

unde:

x1 – coordonata x a primului punct;

y1 – coordonata y a primului punct;

x2 – coordonata x a punctului doi;

y2 – coordonata y a punctului doi;

z1 – coordonata z a primului punct;

z2 – coordonata z a punctului doi.

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 modificată folosind funcția **strokeWeight()**.

Sintaxa:

point(x, y, [z]);

point(coordinate_vector);

unde:

x – coordonata x;

y – coordonata y;

z – coordonata z (în regimul WebGL) (opțional);

coordinate_vector – un vector de coordonate.

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

Sintaxa:

quad(x1, y1, x2, y2, x3, y3, x4, y4);

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

unde:

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 a primului punct;
z2 – coordonata z a punctului doi;
z3 – coordonata z a punctului trei;
z4 – coordonata z a 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 (tl), din dreapta sus (tr), din dreapta jos (br) și, respectiv, din stânga jos (bl). Parametrul razei colțului este setat la valoarea razei specificate anterior în lista de parametri.

Sintaxa:

rect(x, y, w, h, [tl], [tr], [br], [bl]);

rect(x, y, w, h, [detailX], [detailY]);

unde:

x – coordonata x a dreptunghiului;

y – coordonata y a dreptunghiului;

w – lățimea dreptunghiului;

h – înălțimea dreptunghiului;

tl – raza de rotunjire a colțului stânga-sus (opțional);

tr – raza de rotunjire a colțului dreapta-sus (opțional);

br – raza de rotunjire a colțului dreapta-jos (opțional);

bl – raza de rotunjire a colțului stânga-jos (opțional);

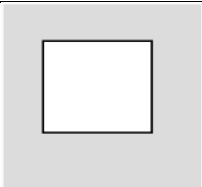
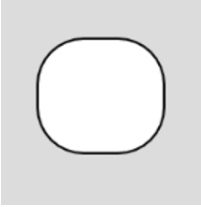
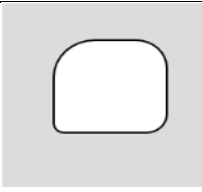
detailX (număr întreg) – numărul de segmente pe axa X (pentru regimul WebGL) (opțional);

detailY (număr întreg) – numărul de segmente pe axa Y (pentru regimul WebGL) (opțional).

Dacă razele de rotunjire a tuturor colțurilor sunt egale ($tl= tr= br= bl$), atunci poate fi indicat doar un singur parametru.

Exemple de utilizare sunt date în tabelul 1.2.

Tabelul 1.2. Exemple de utilizare a funcției rect()

<i>Programul</i>	<i>Rezultatul</i>
<pre>function setup() { createCanvas(100, 100); } function draw() { background(220); rect(20, 20, 55, 50); }</pre>	
<pre>function setup() { createCanvas(100, 100); } function draw() { background(220); rect(20, 20, 55, 50, 20); } // echivalent cu //rect(20, 20, 55, 50, 20,20,20,20);</pre>	
<pre>function setup() { createCanvas(100, 100); } function draw() { background(220); rect(25, 20, 55, 50, 20, 15, 10, 5); }</pre>	

Funcția square(): desenează un pătrat pe ecran, cu fiecare colț de nouăzeci de grade. Această funcție este un caz particular al funcției **rect()** în care lățimea și înălțimea sunt aceleași, iar parametrul s – dimensiunea laturii. În mod implicit, primii doi parametri – coordonata colțului din stânga sus, al treilea – dimensiunea laturii pătratului.

Parametrii patru, cinci, șase și șapte, 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, [t1], [tr], [br], [bl]);**

unde:

x – coordonata x a pătratului;

y – coordonata y a pătratului;

s – mărimea laturii pătratului;

t1 – raza colțului stânga-sus (opțional);

tr – raza colțului dreapta-sus (opțional);

br – raza colțului dreapta-jos (opțional);

bl – raza colțului stânga-jos (opțional).

Funcția triangle(): desenează un triunghi. Argumentele funcției specifică coordonatele primului vârf, coordonatele vârfului doi, iar ultimele două argumente indică coordonatele vârfului al treilea al triunghiului.

Sintaxa:**triangle(x1, y1, x2, y2, x3, y3);**

unde:

x1 – coordonata x a primului punct;

y1 – coordonata y a primului punct;

x2 – coordonata x a punctului doi;

y2 – coordonata y a punctului doi;

x3 – coordonata x a punctului trei;

y3 – coordonata y a punctului trei.

1.2. Atribute grafice

Caracteristicile figurilor geometrice, cum ar fi culoarea figurii, grosimea și culoarea liniei, tipul hașurării, modul de conexiune a liniilor reprezintă atributele grafice simple. În biblioteca p5.js există o listă de funcții care permit setarea sau modificarea acestor atribute.

Principalele atributele grafice sunt descrise în continuare:

Funcția fill(): setează culoarea folosită pentru umplerea figurilor, toate figurile desenate după această funcție vor fi umplute (colorate) cu culoarea indicată de parametrul 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);
```

unde:

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

v2 (număr întreg) – verde sau valoarea 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) – (opțional);

value (un string) – string care indică culoarea;

gray (număr întreg) – valoarea culorii sure;

values (numere []) – un tabel care conține componentele culorilor: roșii, verde, albastră și alpha.

Dacă dorim ca figura să nu fie umplută cu culoare (să fie transparentă) atunci utilizăm funcția **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);

unde:

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

v2 (număr întreg) – verde sau valoarea 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) – (opțional) este utilizat pentru setarea transparenței;

value (un string) – string care indică culoarea;

gray (număr întreg) – valoarea nuanței culorii sure;

values (numere []) – un tabel care conține componentele culorilor: roșu, verde, albastru și alpha.

Dacă este necesar ca figura să fie desenată fără margini, atunci utilizăm funcția **noStroke()**.

În afara culorii, funcția stroke poate specifica următoarele:

strokeCap();

strokeJoin();

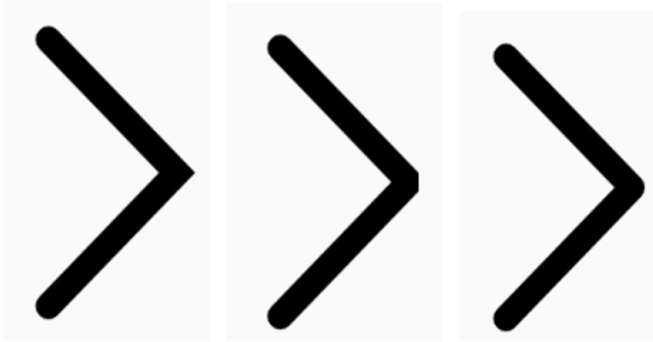
strokeWeight();

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, fie pătrate sau extinse, fiecare dintre acestea fiind specificat cu parametrii corespunzători: ROUND, SQUARE și PROIECT. Implicit este ROUND.

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

Diferența dintre aceste moduri este reprezentată în figura 1.5.



`strokeJoin(MITER);` `strokeJoin(BEVEL);` `strokeJoin(ROUND);`

Figura 1.5. Parametrii funcției strokeJoin

În cazul în care este nevoie de adăugat text, poate fi utilizată funcția `text` care poate avea diferiți parametri.

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 fontul este setat cu funcția `textFont()` și va fi folosită o dimensiune implicită a fontului, dacă nu este setată o altă dimensiune de font cu funcția `textSize()`. Culoarea textului poate fi schimbată cu funcția `fill()`. Schimbarea conturului textului se efectuează cu funcțiile `stroke()` și `strokeWeight()`.

Textul este afișat în dependență de funcția `textAlign()`, care oferă opțiunea de a desena la stânga, la dreapta sau în centrul coordonatelor.

Sintaxa:

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

unde:

str: se indică șirul pentru afișare pe ecran;

x: coordonata x a punctului de început a textului;

y: coordonata y a punctului de început a textului.

Parametrii `x2` și `y2`: definesc o zonă dreptunghiulară în care să se afișeze și pot fi utilizați numai cu date de tip șir 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 x_2 și y_2 nu sunt specificați, alinierea liniei de bază este implicită, ceea ce înseamnă că textul va fi desenat în sus de la x și y .

Informații detaliate pot fi consultate în resursele bibliografice [1], [2] și [4].

1.3. Variabile de sistem în p5.js

O variabilă în p5JS este destinată pentru stocarea informației în memorie și utilizarea ulterioară a acesteia în program. O variabilă poate fi folosită de mai multe ori într-un program, iar valoarea acesteia poate fi modificată în timpul execuției programului.

Pentru crearea variabilelor se folosește următoarea sintaxă:

```
var x;           // Definim variabila x  
x = 12;        // Atribuim valoare variabilei x
```

var – este cuvântul-cheie care indică compilatorului că urmează definirea variabilei.

Variabilele pot fi locale sau globale. Ciclul de viață al variabilei este determinat simplu: variabila este creată în interiorul unui bloc (codul este închis între acolade: `{}`) și există doar în interiorul acestui bloc. *Astfel de variabile sunt numite locale.*

Dacă o variabilă este definită în afara blocurilor (în afara blocului funcției `setup` sau `draw`), *atunci o astfel de variabilă se numește globală.*

În p5js există variabile speciale numite variabile de sistem, care conțin informații despre starea programului în timpul execuției și acestea pot fi accesate de utilizatori:

width – o variabilă care stochează lățimea canvas-ului;

height – o variabilă care stochează înălțimea canvas-ului;

movedX – variabila conține mișcarea orizontală a mouse-ului de la ultimul cadru;

movedY – variabila conține mișcarea verticală a mouse-ului de la ultimul cadru;

mouseX – variabila conține întotdeauna poziția orizontală curentă a mouse-ului în raport cu punctul (0, 0) al pânzei (canvas – ului);

mouseY – variabila conține întotdeauna poziția verticală curentă a mouse-ului în raport cu punctul (0, 0) al pânzei;

pmouseX – variabila de sistem conține întotdeauna poziția orizontală a mouse-ului sau a degetului în cadrul care precede cadrul curent în raport cu punctul (0, 0) al pânzei;

pmouseY – variabila de sistem conține întotdeauna poziția verticală a mouse-ului sau a degetului în cadrul care precede cadrul curent în raport cu punctul (0, 0) al pânzei;

mouseIsPressed – variabila de sistem este adevărată dacă mouse-ul este apăsat, falsă dacă nu;

keyIsPressed – variabila de sistem keyIsPressed este adevărată dacă tasta este apăsată și falsă dacă nu este [3].

1.4. 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 ce determină modul în care datele sunt stocate și afișate pe ecran sau la imprimare.

Formatul fișierului este indicat în numele său ca o parte separată printr-un punct (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, iar fiecare format are propriul algoritm prin care sunt comprimate datele conținute în fișier.

Formatele grafice se clasifică astfel:

- ✓ 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 unui sau altui format

pentru salvarea imaginilor depinde de obiectivele și scopurile de lucru cu imaginea. Dacă acuratețea fotografică a culorilor trebuie să fie calitativă, atunci este preferat unul dintre formatele raster. Pentru sigile, diagrame, elemente de design sunt recomandate formatele 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.

Lucrarea de laborator nr. 1

Tema: SINTEZA IMAGINILOR 2D

Obiectivele lucrării:

1. Familiarizarea cu biblioteca grafică p5.js - înțelegerea și aplicarea conceptelor de bază ale bibliotecii p5.js utilizate pentru crearea scenelor grafice 2D.

2. Utilizarea primitivelor grafice – obținerea cunoștințelor practice privind utilizarea primitivelor grafice simple (puncte, linii, dreptunghiuri, cercuri) pentru a crea o imagine 2D simplă.

3. Crearea scenelor grafice 2D statice - dezvoltarea abilității de a crea și compune scene grafice 2D statice prin combinarea și aranjarea eficientă a primitivelor grafice.

4. Înțelegerea și aplicarea conceptelor de sinteză grafică - exersarea principiilor de sinteză grafică utilizate în construirea scenelor vizuale în grafica 2D.




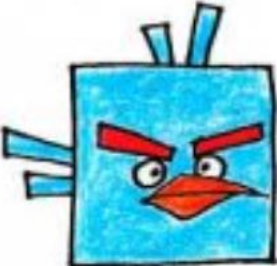


Numărul de ore necesare pentru realizare – 4 ore academice.



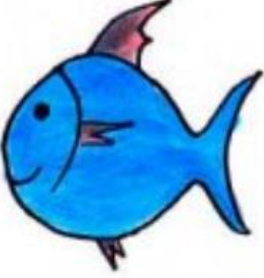




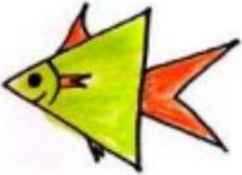
Scopul lucrării: dobândirea cunoștințe practice privind crearea și manipularea scenelor grafice 2D statice, utilizând primitivele grafice oferite de biblioteca p5.js. Aplicarea cunoștințelor teoretice în sinteza imaginilor grafice, utilizarea eficientă a funcțiilor bibliotecii p5.js și crearea imaginii grafice 2D simple.

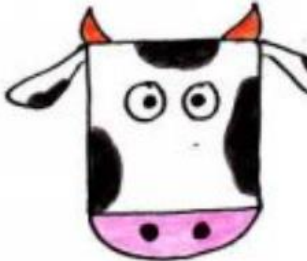





Sarcina lucrării: elaborați un program pentru sinteza unei scene 2D statice utilizând cel puțin 6 primitive grafice de diferite tipuri cum ar fi - *arc()*, *ellipse()*, *circle()*, *line()*, *point()*, *quad()*, *rect()*, *square()*, *triangle()*. Primitivele trebuie să aibă diferite atribute, lucrarea trebuie semnată (numele, prenumele, grupa) în colțul dreapta de jos al ecranului. În program creați personajul conform variantei indicate de profesor, luând în considerație cerințele indicate mai sus. Variantele sunt incluse în tabelul 1.3. Pentru crearea acestor imagini pe pași poate fi consultată pagina:

<https://ja-rastu.ru/raskraski/uroki/1079-poshagovoe-risovanie-zhivotnyh-iz-geometriceskikh-figur.html>.

Tabelul 1.3. Variante pentru realizarea lucrării de laborator

<i>Var.</i>	<i>Personajul</i>	<i>Var.</i>	<i>Personajul</i>
1		11	
2		12	
3		13	

<i>Var.</i>	<i>Personajul</i>	<i>Var.</i>	<i>Personajul</i>
4		14	
5		15	
6		16	
7		17	

<i>Var.</i>	<i>Personajul</i>	<i>Var.</i>	<i>Personajul</i>
8		18	
9		19	
10		20	

Exemplu:

În exemplul ce urmează este reprezentat un cod în care este realizat un personaj în baza condițiilor propuse în sarcina lucrării de laborator.

```
function setup() {
  createCanvas(400, 400);
  background(220); }
function draw() {
// Desenăm capul
  fill(255, 204, 153);
```

```

// colorăm în culoare pielei
    circle(200, 150, 80);

// Desenăm ochii
    fill(0); // negru
    ellipse(185, 145, 10, 10);
    ellipse(215, 145, 10, 10);

// Desenăm gura
    fill(255, 0, 0); // roșu
    arc(200, 165, 30, 20, 0, PI);

// Desenăm corpul
    fill(0, 0, 255); // albastru
    rect(170, 190, 60, 100);

// Desenăm mâinile
    fill(255, 204, 153);
    ellipse(140, 220, 20, 20);
    line(140, 220, 170, 220);
    ellipse(260, 220, 20, 20);
    line(230, 220, 260, 220);

// Desenăm picioarele
    fill(0);
    rect(180, 290, 10, 50);
    rect(210, 290, 10, 50);

// Scriem numele, prenumele și grupa în colțul din dreapta
//de jos
    fill(0);
    textSize(12);
    text("Nume Prenume, grupa", width - 160, height
- 10); }

```


Rezultatul execuției programului este arătat în figura 1.6.

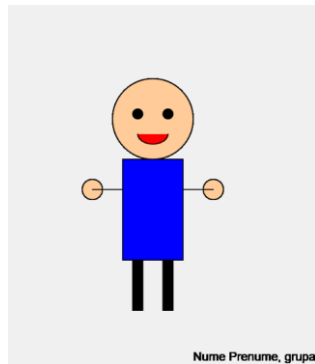


Figura 1.6. Rezultatul execuției codului

Criterii de evaluare

1. Corectitudinea codului (10%) – verificarea corectitudinii codului, fără erori de sintaxă și funcționare.

2. Utilizarea primitivelor grafice (10%) – evaluarea utilizării corecte și varietatea primitivelor grafice (linii, cercuri, pătrate etc.) în conformitate cu cerințele lucrării.

3. Creativitatea și complexitatea compoziției (10%) – măsurarea gradului de creativitate și complexitate a imaginii generate, inclusiv modul în care primitivele sunt combinate pentru a crea o compoziție interesantă.

4. Respectarea instrucțiunilor și cerințelor (10%) – verificarea respectării de către student a cerințelor lucrării, cum ar fi anumite forme sau dimensiuni specifice pentru primitive.

5. Optimizarea codului (10%) – evaluarea eficienței codului în utilizarea resurselor și evitarea codului redundant.

6. Interactivitate (10%) – (dacă este aplicabil) - dacă lucrarea include cerințe interactive, cum ar fi răspunsurile la mouse sau tastatură, se poate evalua dacă aceste funcții sunt implementate corect.

7. Estetică vizuală (10%) – analiza atractivității vizuale a compoziției, cum ar fi armonia culorilor, proporțiile și echilibrul grafic.

8. Respectarea termenului de predare (10%) – evaluarea punctajului în funcție de punctualitate, dacă lucrarea a fost predată în termenul stabilit.

9. Evaluarea cunoștințelor (20%) – explicații privind procesul de realizare a lucrării, ceea ce poate include descrierea funcțiilor principale și a logicii utilizate.

Pentru obținerea notei 5 sunt obligatorii criteriile 1-5, criteriile de evaluare 6-8 oferă câte un punct, iar criteriul 9 – 2 puncte.

Întrebări de autoevaluare a cunoștințelor

1. Enumerați primitivele 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 formatele standard pentru imagini.
5. Enumerați primitivele grafice simple utilizate frecvent în bibliotecile grafice.
6. Ce primitive grafice pot fi utilizate pentru a crea forme de bază într-o scenă grafică 2D?
7. Ce funcții sunt folosite pentru a schimba culoarea și/sau grosimea liniei sau colorarea figurilor grafice?
8. Cum poate fi setată transparența sau stilul de afișare al primitivelor grafice?
9. Explicați cum se utilizează funcțiile pentru contur și umplere în p5.js.
10. Ce funcții sunt utilizate pentru afișarea textului în cadrul unei scene grafice?
11. Cum poate fi modificată dimensiunea, fontul și culoarea textului afișat în p5.js?
12. Explicați cum poate fi poziționat textul într-o scenă grafică.
13. Enumerați câteva formate uzuale pale imaginilor raster.

14. Care sunt diferențele dintre formatele JPEG, PNG și GIF?
15. Ce format de imagine este cel mai potrivit pentru transparență și de ce?

II. GRAFICA VECTORIALĂ

Grafica vectorială este un tip de grafică pe calculator care utilizează descrierea matematică a formelor geometrice (linii, cercuri, dreptunghiuri, poligoane etc.), spre deosebire de grafica raster, care se bazează pe pixeli. Aceasta este folosită pentru a crea imagini care pot fi scalate sau transformate fără pierderea calității, deoarece nu sunt bazate pe o grilă fixă de pixeli.

2.1. Caracteristicile graficii vectoriale

1. Independența de rezoluție: imaginile vectoriale pot fi redimensionate, păstrându-și claritatea și detaliile. Fie că sunt mărite sau micșorate, ecuațiile matematice care stau la baza graficii vectoriale se vor recalcula, deci, imaginile sunt "redesenate", astfel nu vor exista pierderi în calitate, ceea ce le face ideale pentru tipărire și pentru utilizarea în diferite dispozitive cu rezoluții variate.

2. Componentele geometrice: grafica vectorială este construită din obiecte geometrice definite prin formule matematice. De exemplu, un cerc este definit prin raza și coordonatele centrului său, o linie prin punctele de început și sfârșit, iar un poligon prin coordonatele colțurilor sale.

3. Fișierele de dimensiuni mici: deoarece grafica vectorială este descrisă prin formule matematice, ele ocupă un spațiu mai mic în fișierele fizice comparativ cu imaginile raster (care sunt formate dintr-o rețea de pixeli). Acest lucru le face eficiente pentru stocare și partajare.

4. Editabilitatea: obiectele vectoriale pot fi editate individual fără a afecta alte părți ale imaginii. De exemplu, un cerc poate fi modificat fără a afecta alte elemente ale unei scene grafice, ceea ce face ca lucrul cu grafica vectorială să fie mai flexibil decât în cazul graficii raster.

5. Domeniile de aplicare: grafica vectorială este folosită pentru crearea de logo-uri, iconițe, ilustrații, în tipografie și desene tehnice. Un alt domeniu este designul grafic – crearea de materiale

publicitare, afișe, pliante și alte produse de design grafic. Grafica vectorială este utilizată pentru a crea animații, deoarece aceasta permite mișcarea și scalarea ușoară a elementelor grafice.

Avantajele graficii vectoriale:

- **Scalabilitate:** fără pierderi de calitate la redimensionare.
- **Editabilitate:** ușor de modificat și de personalizat fiecare element.
- **Dimensiune mică a fișierelor:** mai eficiente pentru stocare și partajare.
- **Calitate constantă:** imaginile nu devin pixelate indiferent de dimensiunea lor.

Dezavantaje:

- **Complexitate:** crearea unor imagini foarte detaliate poate necesita un timp mai îndelungat de lucru.
- **Limitări în detaliile fine:** graficile vectoriale sunt mai potrivite pentru forme clare și precise, dar pot fi mai greu de folosit pentru imagini cu detalii fine sau efecte complexe de lumină și textură.

2.2. Formatul SVG

Formatul fișierilor grafice vectoriale SVG – Scalable Vector Graphics – este utilizat pentru descrierea imaginilor bidimensionale utilizând XML. SVG este un standard al organizației W3C a cărui proiectare a început în anul 1999. Acest format permite definirea imaginilor prin trei metode: text, grafică vectorială și "bitmap-uri" (harta de biți – fișiere în formatul BMP). Deși există aplicații specializate pentru crearea și editarea de SVG-uri, în acest scop poate fi utilizat orice editor textual. Vizualizarea unei imagini vectoriale SVG poate fi realizată prin utilizarea oricărui browser modern.

În prezent, SVG-ul are mai multe versiuni pentru a se adapta mai bine la diferite cerințe hardware. Astfel, versiunile "SVG Tiny" și "SVG Basic" au fost create special pentru dispozitivele mobile cu resurse limitate. În același timp, profilul "SVG Print" este destinat mediilor de printare a documentelor.

Pentru animarea unei imagini SVG, organizația W3C recomandă standardul "SMIL". Pe lângă recomandarea oficială mai există și alte soluții, cum ar fi "ECMAScript".

Elementul HTML <svg> reprezintă un container pentru grafică vectorială SVG.

În figura 2.1 sunt specificate versiunile browserelor care acceptă complet elementul <svg>.






Element					
<svg>	4.0	9.0	3.0	3.2	10.1

Figura 2.1. Versiunile browserelor care acceptă <svg>

2.3. Primitive grafice 2D în <svg>

Tagul <svg> dispune de câteva metode pentru a desena linii, poligoane, dreptunghiuri, cercuri, text și imagini grafice.

Formatul SVG dispune de câteva elemente pentru descrierea unor forme predefinite care pot fi utilizate de către dezvoltator:

- ✓ **Dreptunghi** <rect>
- ✓ **Cerc** <circle>
- ✓ **Elipsă** <ellipse>
- ✓ **Linie** <line>
- ✓ **Polilinie** <polyline>
- ✓ **Poligon** <polygon>
- ✓ **Cale** <path>

Elementul <line> este utilizat pentru a desena un segment de dreaptă:

Atribute:

x1 – abscisa x a punctului de început al segmentului de dreaptă;
y1 – ordonata y a punctului de început al segmentului de dreaptă;

x2 – abscisa x a punctului de sfârșit al segmentului de dreaptă;
y2 – ordonata y a punctului de sfârșit al segmentului de dreaptă.

Exemplu de utilizare:

```
<line x1="0" y1="0" x2="200" y2="200"  
stroke="black" stroke-width="2" />
```

Formatul SVG oferă un spectru larg de proprietăți pentru contur:

stroke – culoarea liniei, conturului textului ori a figurilor;

stroke-width – grosimea liniei, conturului textului ori a conturului unui element.

Modul de utilizare a diferitor linii este reprezentat în exemplul de mai jos:

```
<svg height="80" width="300">  
<g fill="none">  
<line stroke="red" stroke-width="1", x1="0"  
y1="10" x2="200" y2="10"/>  
<line stroke="black" stroke-width="2", x1="0"  
y1="20" x2="200" y2="20"/>  
<line stroke="blue" stroke-width="3", x1="0"  
y1="30" x2="200" y2="30"/>  
</g>  
</svg>
```

Rezultatul execuției acestui cod este arătat în figura 2.2.



Figura 2.2. Definirea liniilor de culori și grosimi diferite

stroke-linecap – atributul specifică stilul extremelor liniei sau conturului. Există trei opțiuni principale pentru acest atribut:

- **butt**: capătul liniei este drept și se termină exact la coordonata specificată, este implicit;

- **round**: capătul liniei este rotunjit și extins cu o jumătate din lățimea liniei;
- **square**: capătul liniei este drept, dar se extinde dincolo de coordonată, adăugând o lungime egală cu jumătate din grosimea liniei.

Un exemplu de folosire a diferitor atribute ale parametrului `stroke-linecap` și rezultatul execuției acestui cod este arătat în figura 2.3.

Exemplu:

```
<svg height="80" width="300">
  <g fill="none" stroke="black" stroke-
width="10">
    <line stroke-linecap="butt", x1="10" y1="20"
x2="200" y2="20"/>
    <line stroke-linecap="round", x1="10" y1="40"
x2="200" y2="40" />
    <line stroke-linecap="square", x1="10" y1="60"
x2="200" y2="60" />
  </g>
</svg>
```



Figura 2.3. Tipurile atributului stroke-linecap

Atributul stroke-dasharray este utilizat pentru crearea liniilor punctate; valoarea lui constă într-o listă de numere care definesc lungimea liniuțelor și a spațiilor. De exemplu, **stroke-dasharray="5,5"** va crea o linie cu segmente de 5 unități urmate de spații de 5 unități.

Exemplu:

```
<svg height="80" width="300">
  <g fill="none" stroke="black" stroke-width="4">
    <line stroke-dasharray="5,5", x1="10" y1="20"
x2="200" y2="20" />
    <line stroke-dasharray="10,10", x1="10" y1="40"
x2="200" y2="40" />
    <line stroke-dasharray="20,10,5,5,5,10",
x1="10" y1="60" x2="200" y2="60" />
  </g>
</svg>
```

Rezultatul execuției acestei secvențe de cod este arătat în fig.2.4.

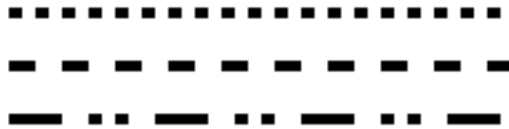


Figura 2.4. Tipuri de linii punctate

Elementul <polyline> este utilizat pentru a crea linii frânte construite din segmente de drepte conectate prin câteva puncte.

Atribut:

points – (o listă de coordonate x, y) conține lista de perechi de coordonate (x, y) necesare pentru desenarea liniei frânte.

Exemplu:

```
<svg height="200" width="500">
  <polyline points="20,20 40,25 60,40 80,120
120,140 200,180" style = "fill: none; stroke:
black; stroke-width:3" />
</svg>
```

Rezultatul rulării acestei secvențe de program este reprezentat în figura 2.5.



Figura 2.5. Definirea liniei frânte

Elementul <circle> definește un cerc.

Atributele:

cx – coordonata pe axa x a centrului cercului;

cy – coordonata pe axa y a centrului cercului;

r – raza cercului.

Exemplu:

```
<svg width="100" height="100">  
  <circle cx="50" cy="50" r="40" stroke="green"  
stroke-width="4" fill="yellow" />  
</svg>
```

Rezultatul rulării acestei secvențe de program este reprezentat în figura 2.6.



Figura 2.6. Definirea cercului în <svg>

Elementul <ellipse> se utilizează pentru a desena o elipsă.

Atributele:

cx – coordonata pe axa x a centrului elipsei;
cy – coordonata pe axa y a centrului cercului;
rx – lungimea razei elipsei pe axa x (raza orizontală – lățimea);
ry – lungimea razei elipsei pe axa y (raza verticală – înălțimea).

Elipsa reprezintă un caz generalizat al cercului. Diferența dintre aceste figuri constă în faptul că elipsa are lungimi diferite ale razelor pe axele x și y, iar în cazul cercului acestea sunt egale.

Exemplu:

```
<svg height="140" width="500">  
<ellipse cx="200" cy="80" rx="100" ry="50"  
tyle="fill:yellow;stroke:purple;stroke-width:  
2" />  
</svg>
```

Rezultatul rulării acestei secvențe de program este reprezentat în figura 2.7.

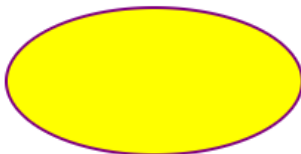


Figura 2.7. Definirea unei elipse

Elementul <rect> permite crearea dreptunghiului și figurilor derivate de la acesta.

Atributele:

x – coordonata pe axa x a colțului din partea stângă de sus a dreptunghiului;

y – coordonata pe axa y a colțului din partea stângă de sus a dreptunghiului;

rx – raza pe axa x (pentru rotunjirea vârfurilor elementului);

ry – raza pe axa y (pentru rotunjirea vârfurilor elementului);

width – lățimea dreptunghiului;

height – înălțimea dreptunghiului.

Exemplu:

```
<svg width="400" height="180">  
  <rect x="50" y="20" rx="20" ry="20" width="150  
height="150"  
  style="fill:red;stroke:black;stroke-  
width:5;opacity:0.5"/>  
</svg>
```

Rezultatul rulării acestei secvențe de program este reprezentat în figura 2.8.

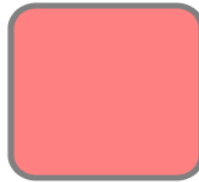


Figura 2.8. Definierea unui dreptunghi

Elementul <polygon> este utilizat pentru crearea elementelor grafice care trebuie să conțină cel puțin 3 laturi. Poligoanele sunt formate din linii drepte, iar figura reprezintă un contur închis.

Atributele:

points – conține coordonatele x și y ale fiecărui punct din mulțimea punctelor care constituie vârfurile poligonului;

Notă: numărul total al punctelor trebuie să fie par;

fill-rule – reprezintă un atribut care definește algoritmul utilizat pentru reprezentarea interiorului figurii (metoda de umplere/colorare) și poate lua valorile evenodd sau nonzero.

Exemplu:

```
<svg height="250" width="500">  
  <polygon points="220,10 300,210 170,250  
123,234" style="fill:lime;stroke:purple;stroke-  
width:1" />  
</svg>
```

Rezultatul rulării acestei secvențe de program este reprezentat în figura 2.9.

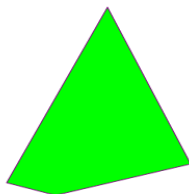


Figura 2.9. Definirea unui poligon cu patru laturi

Elementul <path> este utilizat pentru a defini o cale.

Atributele:

d – parametrul path data reprezintă un set de comenzi care definește calea;

pathLength – dacă este prezent, calea este scalată astfel încât lungimea calculată a căii care trece prin puncte să fie egală cu această valoare;

transform – lista de transformări.

Parametrul path data poate lua valorile:

- ✓ **M** = moveto
- ✓ **L** = lineto
- ✓ **H** = horizontal lineto
- ✓ **V** = vertical lineto
- ✓ **C** = curveto
- ✓ **S** = smooth curveto
- ✓ **Q** = quadratic Bézier curve
- ✓ **T** = smooth quadratic Bézier curveto
- ✓ **A** = elliptical Arc
- ✓ **Z** = closepath

Toate comenzile de mai sus pot fi scrise și cu minuscule. Utilizarea majusculilor semnifică poziționarea absolută, utilizarea minusculilor semnifică poziționarea relativă.

Curbele Bézier sunt utilizate pentru modelarea liniilor netede care pot fi scalate la infinit fără pierderea calității. Pentru reprezentarea curbei Bézier utilizatorul trebuie să definească două puncte care indică cele două capete ale curbei și unul sau două puncte de control. O curbă Bézier cu un punct de control poartă numele de

curbă Bézier pătratică, iar cele cu două puncte de control – curbe Bézier cubice.

Elementul <text> este utilizat pentru a defini un text.

Atributele:

x – lista de coordonate pe axa *x* a caracterelor.

Elementul *n* din lista de coordonate de pe axa *x* corespunde caracterului cu numărul de ordine *n* din text. Dacă există caractere suplimentare cărora nu le corespund valori pentru coordonata de pe axa *x*, acestea sunt plasate după ultimul caracter. Valoarea implicită este 0;

y – lista de coordonate pe axa *y* a caracterelor;

dx – lista valorilor pe axa *x* cu care se deplasează caracterele relative la poziția absolută a ultimului caracter desenat;

dy – lista valorilor pe axa *y* cu care se deplasează caracterele relative la poziția absolută a ultimului caracter desenat;

rotate – lista unghiurilor de rotație.

Valoarea *n* din lista de rotație se aplică caracterului *n* din textul afișat. Caracterele suplimentare nu se rotesc.

2.4. Integrarea graficii vectoriale în p5.js Web Editor

Pentru integrarea codului SVG în limbajul HTML cu ajutorul editorului web p5.js este nevoie de logarea cu ajutorul contului Google, Account sau GitHub pe platforma respectivă. Pentru vizualizarea fișierelor, din componența proiectului trebuie accesat butonul cu pictograma de săgeată din partea stângă a ferestrei, după cum este arătat în figura 2.10, și se deschide fereastra Sketch Files în care se pot vedea fișierele componente ale proiectului: index.html, sketch.js și style.css. Fișierul index.html conține codul HTML pentru marcarea interfeței componentelor paginii web a proiectului. Adăugarea codului SVG în proiect se face prin editarea fișierului index.html, și anume, înlocuirea tag-ului par `<script src="sketch.js"></script>` din interiorul tag-ului `<body></body>` cu tagul par `<svg></svg>` necesar (figura 2.10).

În exemplul de mai jos este reprezentată crearea curbei Bézier pătratică, unde punctele **A** și **C** reprezintă vârfurile, iar punctul **B** reprezintă punctul de control:

Vizualizarea rezultatului execuției programului realizat poate fi observată în partea dreaptă a editorului p5.js Web Editor în fereastra Preview, după cum este arătat în figura 2.10. Dacă imaginea obținută va fi mărită sau micșorată, se poate observa că calitatea acesteia nu se modifică.

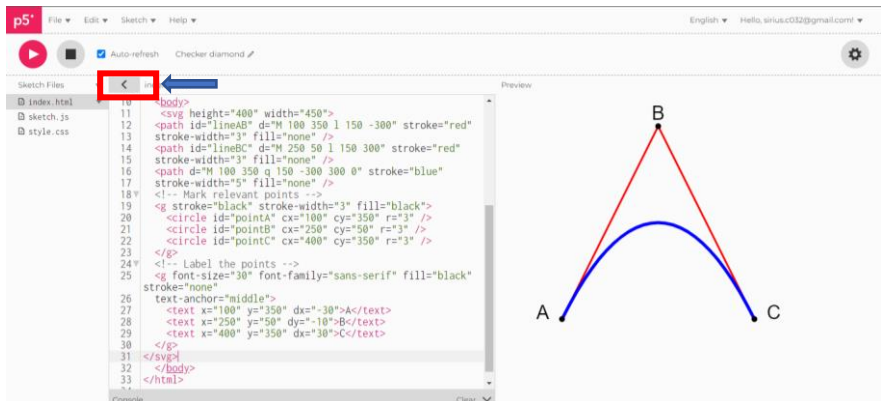


Figura 2.10. Fereastra de lucru a editorului p5.js

2.5. Software gratuit pentru grafica vectorială

Inkscape este un software de editare grafică pentru Windows, Mac OS X și Linux, este gratuit și open source.

Gravit Designer Online este o aplicație online gratuită pentru design și editare vectorială pentru Mac, Win, Linux, Chrome OS, Browser. Aceasta permite editarea vectorială, online, direct în browser.

SVG Viewer este un instrument software care permite vizualizarea, editarea sau manipularea online a fișierelor SVG (Scalable Vector Graphics). SVG Viewer este accesibil pe adresa <https://www.svgviewer.dev>, este un software de editare grafică gratuit folosit pentru crearea graficii vectoriale ușor și intuitiv. Este simplu și potrivit atât pentru web, cât și pentru platformele desktop.

Este un editor vectorial gratuit care poate fi rulat pe un computer personal sau browser și este folosit pentru a crea vectori și alte elemente grafice. Este o multiplatformă ușor de utilizat pe Mac, Windows, Linux, Chrome OS, Browser.

Vectr este un instrument utilizat pe scară largă pentru adnotarea, editarea imaginilor și desenarea machetelor și diagramelor. Programul permite crearea desenelor vectoriale scalabile, utilizând funcții simple și clare. Este ușor de învățat, perfect pentru cei care încep să se familiarizeze cu grafica vectorială și au suficiente capacități de bază de ilustrare.

Software-ul Vectr dispune de un număr mare de tutoriale interactive. Vectr permite crearea și editarea graficii vectoriale, utilizând instrumentele de pe masa de lucru. Există mai multe ghiduri video pe Youtube <https://youtu.be/CnSRzM91FYY>.

Fereastra de lucru a pachetului Vestr este arătată în figura 2.11.

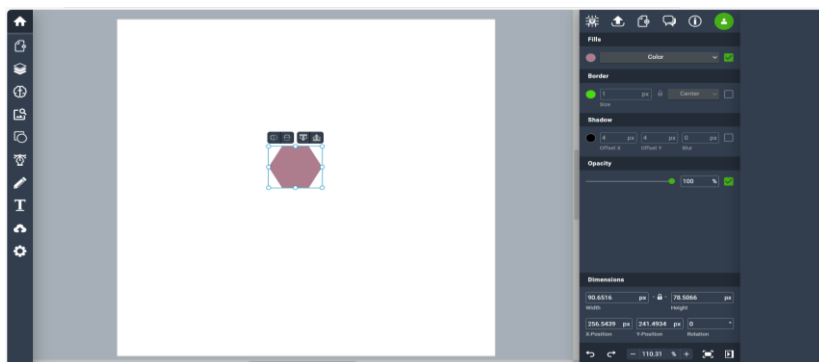


Figura 2.11. Fereastra de lucru a editorului Vectr

Pentru începerea lucrului cu editorul online Vectr, se încarcă o imagine în Vectr. Este permisă importarea fișierelor în formatele EPS, AI, SVG, PNG sau JPEG. Interfața Grafică Utilizator este organizată convenabil. Filele „Pagini” și „Straturi” din colțul din stânga de sus oferă control asupra procesului de creare a proiectului. Panoul de filtrare este situat în partea dreaptă a interfeței. Aplicația oferă opțiuni de schimbare a unghiului, adăugarea umbrelor, marginilor sau diferitor efecte.

Odată cu finalizarea fazei de proiectare, programul oferă opțiunea de a exporta desene în formatul PNG, JPEG sau SVG. SVG este singurul format vectorial scalabil care poate fi deschis în alte aplicații, în timp ce hărțile de biți PNG și JPEG sunt mai potrivite pentru web. Aplicația oferă informații utile și sfaturi practice în secțiunea de instruire. Dezvoltatorii editorului Vectr au oferit utilizatorilor o serie de instrucțiuni și tutoriale pentru rezolvarea celor mai populare sarcini de design: crearea pictogramelor, logo-urilor, tipografiilor, meniurilor, colajelor, infograficilor și multe altele.

Lucrarea de laborator nr.2

Tema: GENERAREA IMAGINILOR VECTORIALE

Obiectivele lucrării:

1. Înțelegerea conceptului de grafică vectorială – familiarizarea cu principiile de bază ale graficii vectoriale și diferențele dintre grafica vectorială și grafica raster.

2. Utilizarea primitivelor grafice SVG – aplicarea primitivelor grafice de bază SVG, cum ar fi: <rect>, <circle>, <ellipse>, <line>, <polyline>, etc.

3. Manipularea atributelor SVG – înțelegerea și aplicarea atributelor corespunzătoare pentru a personaliza aspectul elementelor grafice cum ar fi – fill, stroke, stroke-width, opacity, transform, și altele.

4. Crearea scenei grafice vectoriale 2D – proiectarea și generarea unei scene vectoriale 2D complexe, folosind primitivele menționate pentru a le organiza într-o scenă coerentă.

5. Exportul și vizualizarea imaginilor SVG – înțelegerea procesului de export al imaginilor SVG și vizualizarea acestora în diverse aplicații sau browsere pentru a verifica corectitudinea și calitatea rezultatelor.

6. Obținerea abilităților practice privind integrarea elementelor grafice – combinarea mai multor forme și elemente grafice pentru a construi o scenă completă, dezvoltând abilități privind crearea unor imagini vectoriale mai complexe și atractive.

Numărul de ore necesare pentru realizare – 4 ore academice.

Scopul lucrării: obținerea cunoștințelor practice privind sinteza și generarea scenelor grafice vectoriale 2D, utilizând primitivele grafice simple în formatul SVG. Explorarea conceptelor esențiale precum definirea formelor geometrice și utilizarea atributelor pentru personalizarea acestora.

Sarcina lucrării:

Creați o scenă grafică vectorială 2D, utilizând primitivele grafice necesare cum ar fi: `<rect>`, `<circle>`, `<ellipse>`, `<line>`, `<polyline>`, `<polygon>`, `<path>`, împreună cu atributele corespunzătoare. Scena trebuie să conțină un element `<text>` plasat în colțul de jos, în partea dreaptă a ecranului, care va indica numele, prenumele și grupa studentului. Elaborați versiunea vectorială a personajului desenat în lucrarea de laborator nr.1. Variantele sunt indicate în tabelul 1.3.

Pentru reproducerea formelor sau curbelor complexe este permisă utilizarea editoarelor grafice vectoriale.

Exemplu: crearea imaginii vectoriale bidimensionale care desenează o buburuză.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/p5.js
/1.9.1/p5.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/p5.js
/1.9.1/addons/p5.sound.min.js"></script>
    <link rel="stylesheet" type="text/css"
href="style.css">
```

```

<meta charset="utf-8" />
<style>
  svg {
    filter: drop-shadow(20px 10px 5px
rgba(0,0,0,0.66));
  }
</style>
</head>
<body>
  <svg width="288" height="321" viewBox="0 0 288
321" fill="none" >
    <linearGradient id="grad1" x1="0%" y1="0%"
x2="100%" y2="0%">
      <stop offset="0%" style="stop-
color:rgb(255,255,255);stop-opacity:1" />
      <stop offset="100%" style="stop-
color:rgb(201,16,16);stop-opacity:5" />
    </linearGradient>
    <path d="M235.806 221.399C199.733 279.997
132.807 303.604 86.4958 275.096C40.185 246.587
31.1209 176.2 67.1938 117.602C103.267 59.0032
170.193 35.3956 216.504 63.9043C262.815 92.413
271.879 162.8 235.806 221.399Z"
fill="url(#grad1)" stroke="black" stroke-
width="3"/>
    <ellipse cx="158" cy="250.5" rx="18" ry="15.5"
fill="black"/>
    <ellipse cx="97.5" cy="213.5" rx="13.5" ry="11.5"
fill="black"/>
    <circle cx="213" cy="181" r="18" fill="black"/>
    <circle cx="213" cy="181" r="18" fill="black"/>
    <circle cx="134.5" cy="133.5" r="22.5"
fill="black"/>
    <circle cx="134.5" cy="133.5" r="22.5"
fill="black"/>

```

```

<line x1="167.328" y1="280.303" x2="188.328"
y2="320.303" stroke="black" stroke-width="3"/>
<line x1="44.6949" y1="203.329" x2="0.694881"
y2="226.329" stroke="black" stroke-width="3"/>
<line x1="50.6949" y1="150.329" x2="6.69488"
y2="173.329" stroke="black" stroke-width="3"/>
<line x1="84.2833" y1="93.473" x2="6.28327"
y2="108.473" stroke="black" stroke-width="3"/>
<line x1="254.222" y1="179.131" x2="286.222"
y2="224.131" stroke="black" stroke-width="3"/>
<line x1="223.328" y1="234.303" x2="244.328"
y2="274.303" stroke="black" stroke-width="3"/>
<ellipse cx="212.5" cy="62.5" rx="54.5" ry="41.5"
fill="#1E1E1E"/>
<ellipse cx="191" cy="50.5" rx="10" ry="11.5"
fill="white"/>
<ellipse cx="228" cy="73.5" rx="10" ry="11.5"
fill="white"/>
<ellipse cx="191.5" cy="50.5" rx="3.5" ry="5.5"
fill="black"/>
<ellipse cx="227.5" cy="73.5" rx="3.5" ry="5.5"
fill="black"/>
<ellipse cx="235.5" cy="26" rx="17.5" ry="13"
fill="#1E1E1E"/>
<line x1="227" y1="16" x2="227" stroke="black"
stroke-width="2"/>
<line x1="245.211" y1="25.3861" x2="259.211"
y2="7.38606" stroke="black" stroke-width="2"/>
<line x1="91.5791" y1="274.73" x2="175.579"
y2="143.73" stroke="#1E1E1E"/>
</svg>
</body>
</html>

```

Rezultatul rulării programului este arătat în figura 2.12. Pentru colorarea personajului a fost utilizat gradientul, iar pentru crearea imaginii mai complexe este utilizată umbrirea.

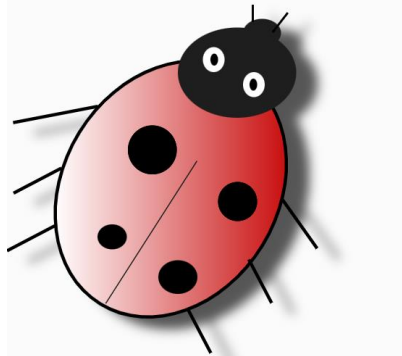


Figura 2.12. Rezultarul execuției programului

Criterii de evaluare

1. Corectitudinea codului și a implementării (30%) – evaluarea corectitudinii utilizării primitivelor grafice SVG, cum ar fi <rect>, <circle>, <ellipse>, <line>, <polyline>, <polygon> și <path>. Verificarea dacă toate primitivele necesare au fost folosite corect, complet și conform cerințelor lucrării, dacă elementele grafice sunt corect desenate și aranjate pentru formarea scenei dorite.

2. Utilizarea corectă a atributelor SVG (20%) – se evaluează aplicarea corectă a atributelor SVG, cum ar fi fill, stroke, stroke-width, opacity, transform și altele pentru personalizarea elementelor grafice. Se verifică dacă modificările stilistice sunt aplicate corespunzător fiecărei forme, respectând cerințele de design.

3. Creativitatea și complexitatea compoziției (15%) – măsurarea gradului de creativitate și complexitate a imaginii generate, inclusiv modul în care primitivele sunt combinate pentru a crea o compoziție interesantă. Se apreciază gradul de originalitate și complexitate al lucrării, adică modul în care studentul adaugă detalii suplimentare pentru a crea o scenă mai interesantă.

4. Estetica și organizarea scenei (10%): – se evaluează aspectul vizual al scenei, inclusiv aranjamentul și compunerea elementelor grafice. Detaliile vizuale sunt adecvate scopului propus și demonstrează o înțelegere a principiilor de desenare.

5. Evaluarea cunoștințelor (20%) – explicații privind procesul de realizare a lucrării, ceea ce poate include descrierea funcțiilor principale și a logicii utilizate.

6. Respectarea termenului de predare (5%) – se va lua în considerare respectarea termenului-limită stabilit de profesor pentru predarea lucrării.

Întrebări de autoevaluare a cunoștințelor

1. Ce reprezintă SVG și cum diferă grafica vectorială de grafica raster?

2. Enumerați primitivele grafice utilizate în SVG și explicați cum se folosește fiecare.

3. Cum este definit un dreptunghi în SVG și care sunt principalele atribute necesare?

4. Explicați rolul atributelor `fill`, `stroke` și `stroke-width`. Cum afectează acestea aspectul primitivei grafice?

5. Ce efect are atributul `opacity` asupra unui element SVG?

6. Cum pot fi modificate dimensiunile unei primitive SVG, cum ar fi un cerc sau dreptunghi, folosind atributele disponibile?

7. Ce avantaje și dezavantaje ale utilizării graficii vectoriale sunt în comparație cu grafica raster?

8. Cum pot fi folosite funcțiile `polyline` și `polygon` pentru a crea forme complexe în SVG?

III. TRANSFORMĂRI GEOMETRICE 2D

Transformările geometrice în 2D reprezintă un set de operații matematice folosite pentru modificarea poziției, dimensiunii și orientării formelor într-un plan bidimensional. Aceste operații sunt esențiale în grafica pe calculator, deoarece permit manipularea obiectelor grafice pentru crearea animațiilor, aplicarea efectelor vizuale și poziționarea elementelor într-o scenă. Principalele transformări geometrice 2D includ translația, rotația, scalarea și reflexia.

3.1 Translația

Translația este procesul de deplasare a unui obiect fără a-i modifica forma, dimensiunea sau orientarea. Aceasta se realizează prin adăugarea unor valori constante la coordonatele fiecărui punct al obiectului.

Translația în p5.js este realizată cu ajutorul funcției **translate()**.

Funcția translate(): deplasează sistemul de coordonate. Implicit, originea sistemului de coordonate se află în punctul cu coordonata (0, 0) (colțul din stânga, sus) al canvas-ului în modul 2D și în centrul canvas-ului în modul WebGL. Tot ce este desenat după ce se apelează translate() va apărea ca fiind deplasat. Există două moduri de a apela translate(), în dependență de parametrii acesteia.

Primul mod de a apela **translate()** folosește numere pentru a seta deplasarea pe axele pozitive x și y. Al treilea parametru, z, este optional, fiind utilizat în grafica 3D.

Al doilea mod de a apela translate() folosește un obiect p5.Vector pentru a seta deplasarea pe axe, în dependență de un vector prestabilit.

În mod implicit, transformările se acumulează. De exemplu, apelarea **translate(10, 0)** de două ori are același efect ca și apelarea **translate(20, 0)** o singură dată.

Notă. Transformările sunt resetate la începutul buclei de desenare.

Sintaxa:**translate(x, y, [Z]);****translate(vector);**

unde:

x – un număr întreg care indică deplasarea pe axa x pozitivă;

y – un număr întreg care indică deplasarea pe axa y pozitivă;

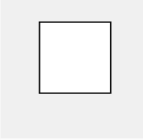
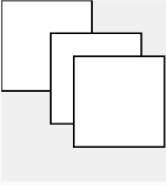

z – un număr întreg care indică deplasarea pe axa z pozitivă;

vector – p5.Vector în baza căruia se realizează deplasarea.

Exemple de utilizare a funcției translate() sunt inserate în tabelul

3.1.

Tabelul 3.1. Exemple de utilizare a funcției translate()

<i>Programul</i>	<i>Rezultatul</i>
<pre> translate(30, 20); rect(0, 0, 55, 55); //echivalent cu //rect(30,20,55,55); </pre>	
<pre> rect(0, 0, 55, 55); //Desenează un dreptunghi începând cu //coordonata 0, 0 translate(30, 20); rect(0, 0, 55, 55); //Desenează un dreptunghi începând cu //coordonata 30, 20 translate(14, 14); rect(0, 0, 55, 55); //Desenează un dreptunghi începând cu //coordonata 44=30+14, 34=20+14 </pre>	
<pre> function draw() { background(200); rectMode(CENTER); translate(width/2, height/2); translate(p5.Vector.fromAngle (millis()/1000, 40)); rect(0, 0, 20, 20); } </pre>	

Mai jos este dat un exemplu de realizare a translației.

Exemplu:

```
let x = 0;
let y = 0;
let dim = 80.0;
function setup() {
  createCanvas(500, 300);
  noStroke();
}
function draw() {
  background(240);
  // Animează creșterea valorii coordonatei x
  x = x + 0.8;
  // Dacă imaginea iese din dimensiunile canvas-ului poziția
  // este resetată
  if (x > width + dim)
  {
    x = -dim;
  }
  // Chiar dacă funcția „rect” desenează forma cu centrul în
  // origine, translația o deplasează în poziție nouă x și y
  translate(x, height / 2 - dim / 2);
  fill(255);
  rect(-dim / 2, -dim / 2, dim, dim);
  // Transformările se acumulează. Observați cum pătratul
  // negru se va mișcă de două ori mai repede decât cel alb
  // chiar dacă are același parametru pentru valoarea axei x
  translate(x, dim);
  fill(0);
  rect(-dim / 2, -dim / 2, dim, dim);
}
```

Rezultatul execuției codului este arătat în figura 3.1.



Figura 3.1. Exemplu de utilizare a funcției translate()

3.2. Rotația

Rotația presupune rotirea unui obiect în jurul unui punct fix (de obicei, originea sistemului de coordonate) cu un anumit unghi. Aceasta modifică orientarea obiectului fără a-i schimba dimensiunea.

Funcția rotate(): realizează transformarea geometrică de rotație. Funcția rotate() schimbă orientarea prin rotirea sistemului de coordonate în jurul originii. Tot ce se desenează după apelarea rotate() va părea că fiind rotit.

Primul parametru indică unghiul cu care se rotește figura ce urmează după apelul funcției rotate(), ce interpretează valorile unghiului fie grade, fie radiani în dependență de setările curente ale funcției **angleMode()**.

Obiectele se rotesc întotdeauna în jurul poziției lor relative față de origine, valorile pozitive rotesc obiectele în sensul acelor ceasornicului. Transformările sunt aplicate funcțiilor ce urmează după, iar apelurile ulterioare ale funcției acumulează efectul. De exemplu, apelarea rotate (HALF_PI), apoi rotate (HALF_PI) este aceeași ca și rotate (PI). Toate transformările sunt resetate când draw() începe din nou.

Din punct de vedere tehnic, rotate() înmulțește matricea de transformare curentă cu matricea de rotație.

Sintaxa:

```
rotate(angle, [axis]);
```

unde:

angle – unghiul de rotire specificat în radiani sau grade, în funcție de unghiul curent setat de angleMode;

axis – (în 3d) setează axa în jurul căruia se va roti figura, este opțional.

Notă. Transformările sunt resetate la începutul buclei de desenare.

Exemple de utilizare și rezultatele execuției codului funcției rotate() sunt incluse în tabelul 3.2

Funcțiile rotateX(), rotateY(), rotateZ(): rotirea în jurul axelor X, Y și Z respectiv.

Sintaxa:

rotateX(angle);

rotateY(angle);

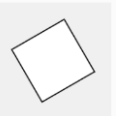

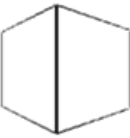
rotateZ(angle);

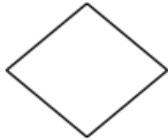
unde:

angle – unghiul de rotație specificat în radiani sau grade, în funcție de modul setat în curentMod.

Exemple de utilizare a funcțiilor sunt date în tabelul 3.2.

Tabelul 3.2. Modul de utilizare a funcției rotate

<i>Programul</i>	<i>Rezultatul</i>
<pre>translate(width / 2, height / 2); rotate(PI / 3.0); rect(-26, -26, 52, 52);</pre>	
<pre>function draw() { background(255); rotateX(millis() / 1000); box();</pre>	
<pre>function draw() { background(255); rotateY(millis() / 1000); box();</pre>	

<i>Programul</i>	<i>Rezultatul</i>
<pre>function draw() { background(255); rotateZ(millis() / 1000); box();</pre>	

În exemplul de mai jos este realizată rotirea dinamică a dreptunghiului în jurul centrului său. Transformarea este realizată aleatoriu la fiecare secundă pară, dând impresia unei vibrații sau a unei mișcări ușoare:

```
let angle = 0.0;
let jitter = 0.0;
function setup() {
  createCanvas(300, 300);
  noStroke();
  fill(255);
  // Este desenat un dreptunghi în centrul canvasului
  // și este realizată rotația în jurul centrului figurii
  rectMode(CENTER); }
function draw() {
  background(220);
  // La fiecare secundă pară se actualizează variabila jitter cu
  //o valoare aleatorie între -0.1 și 0.1. Aceasta creează
  //variație în unghiul de rotație
  if (second() % 2 === 0) {
    jitter = random(-0.1, 0.1); }
  angle = angle + jitter;
  // Se calculează cosinusul unghiului curent pentru o mișcare
  // mai lină, alternând între rotații ușoare în sensul acelor
  //ceasornicului și în sensul invers,
  // când nu este aplicată vibrația
  let c = cos(angle);
  // Figura se deplasează în centrul canvasului
  translate(width / 2, height / 2);
  rotate(c);
```

```
rect(0, 0, 180, 180); }
```

Rezultatul programului este reprezentat în figura 3.2.

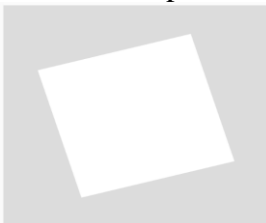


Figura 3.2. Rezultatul rulării programului

3.3. Scalarea

Scalarea – transformarea geometrică care modifică dimensiunea unui obiect prin multiplicarea coordonatelor fiecărui punct cu factor de scalare. Aceasta poate fi uniformă (când factorii de scalare pe axa X și axa Y sunt egali) sau neuniformă (când factorii de scalare sunt diferiți).

Funcția scale() – transformarea care mărește sau micșorează dimensiunea figurii prin extinderea și îngustarea vârfurilor. Obiectele sunt întotdeauna scalate față de originea lor sau față de sistemul de coordonate. Factorii de scalare sunt indicați în procente zecimale. De exemplu, apelul funcției **scale(2.0)** mărește dimensiunea figurii cu 200%.

Transformările sunt aplicate tuturor figurilor apelate după funcția scale. Funcția are efect cumulativ, de exemplu, apelarea scale(2.0), apoi scale(1.0) este aceeași cu scale(3.0), dacă scale() este apelată în draw(), transformarea este resetată când ciclul începe din nou.

Utilizarea acestei funcții cu parametrul z este disponibilă numai în modul WEBGL.

Sintaxa:

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

```
scale(scales);
```

unde:

s – procentul pentru scalarea obiectului sau procentul pentru a scala obiectul de-a lungul axei X dacă sunt date mai multe argumente;

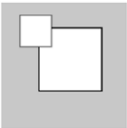
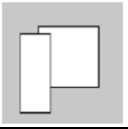
y – procentul pentru scalarea obiectului de-a lungul axei y (opțional);

z – procentul pentru scalarea obiectului de-a lungul axei z (numai WebGL) (opțional);

scales – procentul pe axe pentru scalarea obiectului.

Un exemplu de utilizare a funcției scale este inserat în tabelul 3.3. Dacă în funcția scale indicăm doar un parametru, atunci va fi realizată o scalare uniformă ($sx=sy$), dacă indicăm doi parametri diferiți, atunci scalarea va fi neuniformă.

Tabelul 3.3. Modul de utilizarea a funcției scale()

<i>Programul</i>	<i>Rezultatul</i>
<pre>rect(30, 20, 50, 50); scale(0.5); //uniform rect(30, 20, 50, 50);</pre>	
<pre>rect(30, 20, 50, 50); scale(0.5, 1.3); //neuniform rect(30, 20, 50, 50);</pre>	

În p5.js sunt mecanisme speciale cum ar fi: variabila globală **frameCount** și **millis()**, care sunt folosite pentru a măsura timpul sau pentru a controla secvențele de animație și evenimentele temporizate în scene. Deși ambele măsoară timpul, fiecare are o utilizare specifică:

- **frameCount** – variabilă globală de sistem care numără câte cadre (frame-uri) au fost desenate de la începutul execuției scenei (programului). Aceasta indică numărul de execuții ale funcției draw() din momentul inițializării programului. FrameCount începe cu valoarea 0 în setup() și se incrementează la fiecare execuție completă a funcției draw(), este utilă la crearea animațiilor sau efectelor temporizate, deoarece fiecare valoare reprezintă un moment specific în timp (de la începutul programului);

• **millis()** – funcție ce returnează numărul de milisecunde care au trecut de la începutul programului. Funcția `millis()` urmărește cât timp a rulat o schiță în milisecunde (miimi de secundă). Această informație este adesea utilă pentru sincronizarea evenimentelor și animațiilor.

Funcția `millis()` începe să urmărească timpul înainte ca codul din `setup()` să ruleze. Dacă schița include o funcție `preload()`, `millis()` începe să urmărească timpul imediat ce codul din `preload()` începe să ruleze.

Este utilizată pentru a măsura timpul real și este mai precisă pentru cronometrare sau pentru evenimente independente de numărul de cadre, nu este afectată de fluctuațiile `frameCount` (ratei de cadre), fiind o măsurătoare precisă a timpului trecut.

Este utilizată pentru a crea animații bazate pe timp sau pentru a declanșa evenimente după un anumit interval de timp.

În afara transformărilor geometrice de bază, în `p5.js` există și alte transformări cum ar fi forfecarea, reflexia și compunerea transformărilor.

Funcția `shearX()`: forfecarea sau întinderea deformează o figură în jurul axei `X` cu valoarea specificată de parametrul `angle`. Unghiurile trebuie specificate în `angleMode()` curent. Obiectele sunt întotdeauna tăiate în jurul poziției lor relative față de origine, iar pozitive deformează obiectele în sensul acelor ceasornicului.

Transformările sunt aplicate tuturor funcțiilor apelate după, iar fiecare apel a funcției acumulează efectul. De exemplu, apelarea `shearX(PI/2)`, apoi `shearX(PI/2)` este aceeași cu `shearX(PI)`. Dacă `shearX()` este apelată în `draw()`, transformarea este resetată când ciclul începe din nou.

Din punct de vedere tehnic, `shearX()` înmulțește matricea de transformare curentă cu matricea de rotație. Exemplu de utilizare și rezultatul execuției codului funcției este dat în figura 3.3 (a).

Sintaxa:

`shearX(angle)`;

unde:

`angle` – unghiul de deplasare specificat în radiani sau grade, în funcție de unghiul curent din `angleMod`.

Funcția shearY(): la fel ca și funcția shearX() doar de-a lungul axei Y.

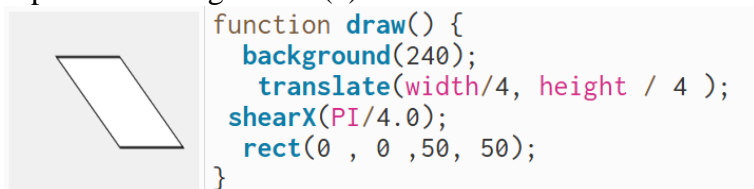
Sintaxa:

shearY(angle);

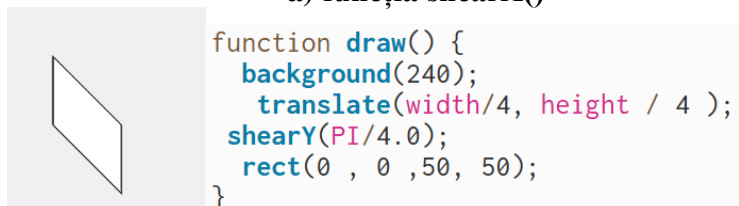
unde:

angle – unghiul de deplasare specificat în radiani sau grade, în funcție de unghiul curent din angleMod.

Un exemplu de utilizare și rezultatul execuției codului funcției este reprezentat în figura 3.3 (b).



a) funcția shearX()



b) funcția shearY()

Figura 3.3. Exemple de utilizare a funcțiilor shear

În p5.js, funcțiile **push()** și **pop()** sunt utilizate pentru a salva sau a restabili setările de stil și transformările grafice dintr-o scenă. Ele permit izolarea transformărilor și a stilurilor aplicate anumitor obiecte pentru a preveni ca acestea să afecteze restul scenei și sunt esențiale pentru a crea scene complexe, fără a avea transformări nedorite asupra întregii scene.

Funcția push(): salvează starea curentă a stilului (culori, grosimea liniilor etc.) și a transformărilor (translații, rotații, scalări) într-o „stivă” temporară. Această funcție poate fi utilizată la începutul unui bloc de cod în care urmează a fi aplicate transformări sau stiluri specifice unui obiect.

Funcția pop(): restaurează starea salvată anterior de **push()**, eliminând orice schimbări de stil sau transformare aplicate după **push()**. Este utilizată la sfârșitul blocului de cod pentru a reveni la setările originale, astfel încât transformările aplicate unui obiect să nu afecteze și alte elemente din scenă.

Aceste funcții sunt utilizate în cazul în care într-o scenă complexă sunt mai multe obiecte, **push()** și **pop()** care ajută să fie menținut controlul asupra transformărilor. Funcțiile permit modificarea doar a anumitor obiecte fără a afecta global coordonatele, culorile sau stilul întregii schițe.

Lucrarea de laborator nr.3

Tema: CREAREA SCENEI DINAMICE 2D

Obiectivele lucrării:

1. Familiarizarea, implementarea și utilizarea transformărilor grafice 2D de bază, utilizând funcțiile **rotate()**, **translate()** și **scale()**.
2. Crearea unei scene dinamice cu obiecte animate, aplicând transformări grafice diverse pentru a modifica poziția, dimensiunea și orientarea elementelor în cadrul scenei grafice.
3. Experimentarea animării transformărilor pentru a genera efecte vizuale și interacțiuni dinamice în scena 2D.
4. Utilizarea bibliotecii **p5.js** pentru implementarea funcționalităților necesare în JavaScript și studierea potențialului acestei biblioteci în grafica pe calculator.

Numărul de ore necesare pentru realizare – 4 ore academice.

Scopul lucrării: dezvoltarea competențelor practice în implementarea și utilizarea transformărilor grafice 2D într-o scenă dinamică, utilizând biblioteca **p5.js**. Dobândirea cunoștințelor practice privind modalitățile de manipulare și animare a obiectelor, folosind funcțiile **rotate()**, **translate()** și **scale()** în **p5.js**.

Sarcina lucrării: utilizând transformările **translate()**, **rotate()** și **scale()**, realizează sarcina individuală.

Varianta 1. Pendul animat: să se creeze un pendul care se mișcă de la stânga la dreapta, simulând mișcarea naturală a unui pendul real folosind funcția **rotate()**. Pendulul trebuie să se rotească în funcție de timp (**millis()** sau **frameCount**).

- **Obiective:** folosirea rotației pentru animarea mișcării pendulului.

- **Sugestie:** să se utilizeze **sin()** pentru o mișcare lentă și naturală.

Varianta 2. Sistemul solar: să se construiască un model simplu al sistemului solar în care planetele orbitează în jurul soarelui folosind **rotate()** și **translate()**. Fiecare planetă trebuie să aibă orbită diferită și mișcare proprie.

- **Obiective:** aplicarea rotațiilor și translațiilor pentru simularea orbitelor planetare.

- **Sugestie:** planetele trebuie să se miște cu viteze diferite.

Varianta 3. Zmeu controlat de mouse: să se creeze o animație în care un zmeu se mișcă pe planul ecranului, iar coarda acestuia se ajustează automat în funcție de poziția mouse-ului. Să se folosească **translate()** și **rotate()** pentru a direcționa zmeul corect în funcție de mișcarea mouse-ului.

- **Obiective:** utilizarea poziției **mouseX** și **mouseY** pentru a controla zmeul.

- **Sugestie:** adaugați mișcări ușor fluide folosind **lerp()** pentru coadă.

Varianta 4. Animarea florii care crește: să se simuleze creșterea unei flori pe ecran. Începe cu o tulpină simplă, adăugând treptat petale care cresc și se rotesc ușor. Folosiți funcțiile **scale()** și **rotate()** pentru a anima petalele.

- **Obiective:** animația creșterii folosind transformări grafice.

- **Sugestie:** controlați viteza de creștere folosind **frameCount**.

Varianta 5. Mozaic interactiv: să se creeze un mozaic de dreptunghiuri care își schimbă culoarea și forma pe măsură ce este

mișcat mouse-ul peste ele. Utilizați **translate()**, **rotate()** și **scale()** pentru a schimba aspectul fiecărui dreptunghi la interacțiunea cu mouse-ul.

- **Obiective:** implementarea interacțiunii bazate pe poziția mouse-ului.

- **Sugestie:** fiecare dreptunghi trebuie să răspundă independent la mișcarea mouse-ului.

Varianta 6. Iluzie optică cu rotații: să se construiască designul grafic care creează o iluzie optică, folosind cercuri sau linii care se rotesc și se scalează continuu. Animația trebuie să fie fluidă și să dea impresia de mișcare circulară.

- **Obiective:** crearea unei iluzii optice prin rotații continue.

- **Sugestie:** folosiți **rotate()** și **scale()** pentru crearea efectelor interesante.

Varianta 7. Fractali cu transformări: să se creeze modelul fractalului în care transformările **rotate()**, **scale()** și **translate()** sunt folosite pentru generarea unui arbore fractal sau o structură geometrică repetitivă. Structura trebuie să fie generată recursiv.

- **Obiective:** folosirea recursivității pentru generarea unui fractal.

- **Sugestie:** fiecare nivel de recursivitate trebuie să micșoreze dimensiunea formelor.

Varianta 8. Cascada de bile: simulați o cascadă de bile care cad pe ecran și ricoșează de la un perete. Fiecare bilă trebuie să se rotească în timp ce cade și să-și schimbe dimensiunea ușor când atinge peretele (efect de rebound).

- **Obiective:** simularea gravitației și a mișcării de bounce folosind rotația și scalarea.

- **Sugestie:** folosiți un array pentru a gestiona mai multe bile simultan.

Varianta 9. Crearea animației interactive: să se creeze o scenă interactivă în care utilizatorul poate controla transformările unui obiect (scalare, rotire, translație) cu mouse-ul și tastatura.

- **Obiective:** utilizarea **mouseX**, **mouseY** pentru controlarea mișcării (translația), folosirea tastelor pentru schimbarea dimensiunii (scalare) și unghiului de rotație.

- **Sugestie:** obiectele trebuie să răspundă la acțiunile utilizatorului în timp real.

Varianta 10. Generarea modelului geometric dinamic: realizați o scenă în care mai multe forme geometrice (cercuri, dreptunghiuri, poligoane) sunt generate și animate pe baza unui set de transformări. Fiecare obiect nou adăugat trebuie să fie scalat și rotit în funcție de poziția sa și să se miște dinamic pe ecran.

- **Obiective:** formele trebuie să fie generate aleatoriu sau pe baza unui pattern matematic (ex.: spirală, grilă).

- **Sugestie:** utilizați **random()** pentru generarea pozițiilor sau dimensiunilor și **frameCount** pentru a anima obiectele pe ecran.

Varianta 11. Ceas analog animat: să se creeze un ceas analog care să funcționeze în timp real folosind transformările **rotate()** și **translate()**. Acele ceasului trebuie să fie animate și să indice ora exactă, minutele și secunde.

- **Obiective:** folosirea **hour()**, **minute()**, **second()** din **p5.js** pentru a determina poziția acelor.

- **Sugestie:** utilizați **rotate()** pentru a anima acele în funcție de timp.

Varianta 12. Simularea gravitației: să se creeze simularea în care mai multe obiecte cad pe ecran sub influența gravitației, iar când ating partea de jos, ele sar în sus (efect de bounce).

- **Obiective:** utilizarea transformărilor grafice pentru scalarea obiectelor când se apropie de partea inferioară (ca și cum ar fi comprimate de sol).

- **Sugestie:** obiectele trebuie să fie translatare în jos în funcție de gravitație. Atingând solul, ele trebuie să fie rotite sau scalate pentru a simula efectul de bounce.

Varianta 13. Creație artistică generativă: realizați o compoziție artistică generativă care folosește transformări multiple (**translate**, **rotate**, **scale**) pentru a genera un model geometric interesant bazat pe un set de reguli simple.

- **Obiective:** folosirea funcției recursive sau buclei pentru a genera structuri complexe.

- **Sugestie:** rezultatul trebuie să fie random de fiecare dată când este rulat.

Exemplu:

Realizați o scenă animată care include cel puțin trei obiecte diferite (forme geometrice), fiecare obiect fiind manipulat prin transformările `translate()`, `rotate()` și `scale()`.

- Toate cele trei obiecte trebuie să fie animate (translarea, rotația și scalarea trebuie să se desfășoare în mod continuu).

- Animația trebuie să fie fluidă și să utilizeze `frameCount` sau `millis()` pentru a controla mișcările:

```
function setup() {
  createCanvas(600, 600);
  noStroke(); }
function draw() {
  background(200);
  // Obiect 1: Dreptunghi care se rotește și se scalează
  push();
  translate(width / 4, height / 2);
  rotate(radians(frameCount % 360));
  scale(sin(frameCount * 0.01) + 1.5);
  // Scalare dinamică
  fill(255, 0, 0);
  rect(-50, -50, 100, 100);
  pop();
  // Obiect 2: Cerc care se mișcă de-a lungul axei X și se
  rotește
  push();
  translate((frameCount % width), height / 4);
  rotate(radians(frameCount % 360));
  fill(0, 255, 0);
  ellipse(0, 0, 80, 80);
  pop();
```

```

// Obiect 3: Triunghi – se mișcă pe diagonală și se scalează
push();
translate((frameCount % width) / 2, (frameCount %
height) / 2);
scale(abs(sin(frameCount * 0.01)) + 0.5); //
Scalare dinamică
fill(0, 0, 255);
triangle(-30, 30, 30, 30, 0, -40);
pop(); }

```

Rezultatul execuției programului este arătat în figura 3.4.

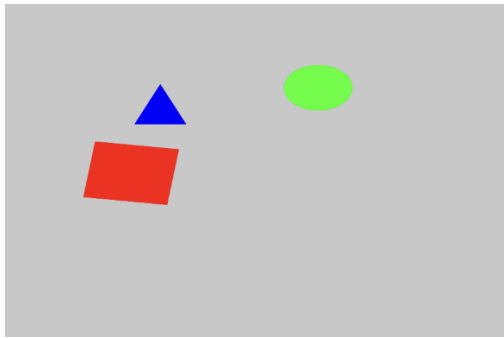


Figura 3.4. Scenă dinamică 2D

Criterii de evaluare

- 1. Implementarea funcțiilor de transformare grafică (30%)**
– utilizarea corectă a funcțiilor `rotate()`, `translate()` și `scale()` în cadrul scenei 2D pentru crearea mișcării și efectelor dinamice ale obiectelor. Implementarea transformărilor creative și corespunzător contextului scenei.
- 2. Fluiditatea și continuitatea animației (10%)** – utilizarea `frameCount` sau `millis()` pentru controlul mișcărilor.
- 3. Complexitatea și diversitatea obiectelor din scenă (10%)**
– modul în care acestea interacționează vizual.
- 4. Originalitatea** la crearea scenei animate și utilizarea eficientă a transformărilor (10%).

5. Corectitudinea codului (20%) – verificarea corectitudinii codului, optimal, fără erori de sintaxă și funcționare.

6. Respectarea termenului de predare (10%) – evaluarea punctajului în funcție de punctualitate, dacă lucrarea a fost predată în termenul stabilit.

7. Evaluarea cunoștințelor (10%) – explicații privind procesul de realizare a lucrării, ceea ce poate include descrierea funcțiilor principale și a logicii utilizate.

Întrebări de autoevaluare a cunoștințelor

1. Ce este o transformare geometrică și de ce este importantă în grafica 2D?

2. Cum influențează funcția `translate()` poziția unui obiect în cadrul unei scene?

3. Ce rol are funcția `rotate()` într-o animație 2D și în jurul cărui punct are loc rotația implicită?

4. Cum funcționează funcția `scale()` și cum afectează aceasta dimensiunea și orientarea obiectelor.

5. Cum pot fi folosite funcțiile `translate()` și `rotate()` împreună pentru a roti un obiect în jurul unui punct specific din scenă?

6. Cum poate fi evitată influența transformărilor asupra tuturor obiectelor din scenă?

7. Cum se utilizează funcțiile `push()` și `pop()` în `p5.js` și de ce sunt necesare când sunt aplicate transformările grafice?

8. Cum se pot aplica simultan mai multe transformări grafice asupra unui obiect?

9. Care este diferența dintre `frameCount` și `millis()` și când ar fi mai util să fie folosită una în locul celeilalte pentru o animație?

10. De ce este important a folosi `push()` și `pop()` când se aplică transformări multiple unui obiect?

11. Cum se poate controla viteza de rotație a unui obiect folosind `frameCount` sau `millis()`?

12. Ce se întâmplă dacă sunt aplicate mai multe transformări fără a folosi `push()` și `pop()`? Cum afectează acest lucru alte obiecte din scenă?

13. Care este diferența dintre `translate()` și `rotate()` în ceea ce privește impactul asupra poziției și orientării unui obiect?

14. Cum se poate crea o animație ciclică folosind funcțiile `sin()` sau `cos()` pentru transformările de scalare sau rotație?

15. Cum pot fi folosite evenimentele mouse-ului (de exemplu, `mouseX` și `mouseY`) pentru a adăuga interactivitate în scenă?

IV. UTILIZAREA OBIECTELOR 3D ÎN SCENE STATICE

4.1. Fișiere *.OBJ

Formatul fișierului 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, fiind utilizat pe scară largă ca format de schimb neutru pentru modelele 3D non-animat în aplicațiile grafice.

Fișierul OBJ este un format standard al imaginilor 3D care poate fi exportat și deschis de diverse editoare 3D. Acesta conține obiecte tridimensionale, care includ coordonatele 3D, hărțile de texturi, fețele poligonale și alte informații despre obiect.

Formatul fișierelor OBJ stochează informații despre modelele 3D, poate codifica geometria suprafeței unui model 3D și poate stoca informații despre culoare și textură. Acest format nu stochează nicio informație despre scenă cum ar fi poziția luminii sau animației.

Fișierul OBJ este generat de un software CAD (Computer Aided Design) ca produs finit al procesului de modelare 3D. Extensia fișierului corespunzătoare formatului fișierului OBJ este – „*.OBJ”.

Formatul fișierelor OBJ este open source și neutru, fiind adesea utilizat la partajarea modelelor 3D în aplicațiile grafice, deoarece asigură un suport bun de import și export din aproape toate software-urile CAD. Acesta este de asemenea utilizat ca formatul fișierului pentru imprimarea 3D multicolor, deoarece formatul standard de imprimare 3D STL nu include informații despre culoare și textură.

Formatul fișierelor OBJ a fost creat inițial de Wavefront Technologies privind aplicația sa Advanced Visualizer pentru a stoca obiecte geometrice compuse din linii, poligoane, 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 fișiere vechi care, deși este foarte popular, nu ține cont de cerințele moderne. Exactitatea imprimării 3D se apropie de rezoluția de nivel de micron și modelele multicolore

devin din ce în ce mai populare. Formatul fișierelor STL nu poate accepta 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 acceptă 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 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 este nevoie de modele cu rezoluție înaltă, multicolor. Pe de altă parte, formatul fișierului OBJ nu este la fel de universal ca și 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.

4.2. Descrierea obiectelor și metodelor principale în lucrul cu fișierele *.OBJ în editorul online p5.js

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

Biblioteca p5.js are un set complet de funcționalități grafice. Cu ajutorul acestei biblioteci 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

Funcția loadModel(): trebuie plasată în interiorul funcției **preload()**, aceasta permite modelului să se încarce complet înainte de a rula programul până la sfârșit.

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, se poate apela **loadModel()** cu parametrul de normalizare setat ca și true. Aceasta va redimensiona modelul la o scară adecvată pentru p5. De asemenea, se pot face modificări suplimentare ale dimensiunii modelului cu funcția **scale()**.

La fel, nu este realizat suportul pentru fișierele colorate STL. Fișierele STL colorate vor fi redatate fără proprietăți de culoare.

Sintaxa:

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

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

unde:

path String – calea modelului încărcat;

normalize boolean – dacă este adevărat, se scalează modelul la o dimensiune standard la încărcare;

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

funcția failureCallback Function(Event) – apelat cu eveniment de eroare dacă modelul nu reușește să se încarce (opțional);

fileType String – extensia fișierului modelului (.stl, .obj) (opțional). Întoarce p5.Geometry – obiect de tip p5.Geometry.

Funcția model() – redă un model 3D pe ecran;

Sintaxa:

```
model(model);
```

unde:

model – p5.Geometrie.

Model 3D încărcat care trebuie redat:

Sketch Files->Upload file

4.3. Crearea scenei statice 3D

Sarcina lucrării de laborator constă în conceperea și construirea scenei 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 desfășurării lucrării de laborator se cere a crea o scenă statică care ar reprezenta o moară de vânt îngrădită de un gard în ograda căreia se vor afla câteva animale, un arbore și un stog de fân. Modelele 3D ale acestor obiecte pot fi create sau importate din repozitoriul 3D Warehouse care poate fi accesat din meniul de bază al editorului grafic 3D Google SketchUp **Window->3D Warehouse**, apoi editate după necesitate, după cum este arătat în figura 4.1.

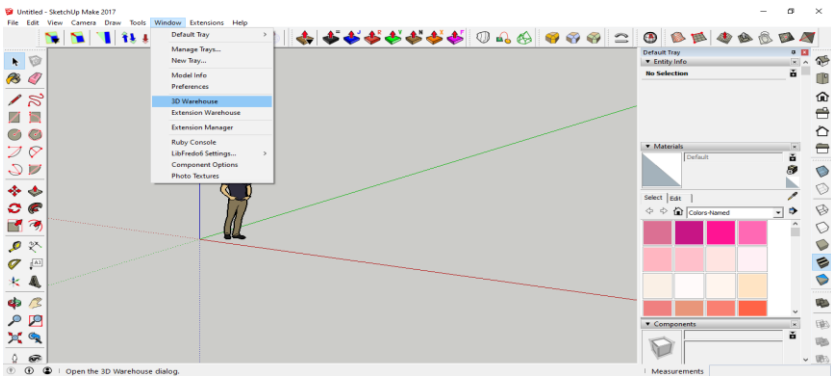


Figura 4.1. Fereastra de lucru a repozitoriului 3D Warehouse

Pentru a 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 intermediul plug-inului **LIPID OBJ Exporter**, care poate fi instalat din meniul **Window->Extension Warehouse**. În figura 4.2, în câmpul de căutare al căruia indicăm cuvântul-cheie OBJ, ca răspuns sunt afișate plugin-urile ce satisfac

criteriile de căutare printre care poate fi observat și plugin-ul căutat *LIPID OBJ Exporter*, după cum este arătat în figura 4.3.

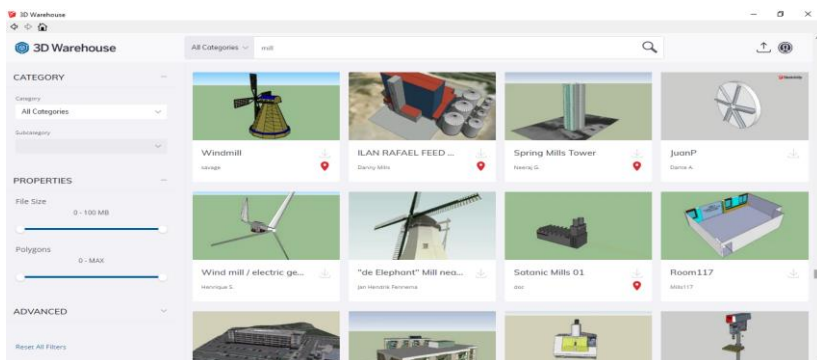


Figura 4.2. Căutarea modelelor 3D în repozitoriul 3D Warehouse

Pentru a instala plugin-urile, utilizatorul trebuie să dispună de un cont Google Account și să fie autentificat în repozitoriul online *Extension Warehouse*.

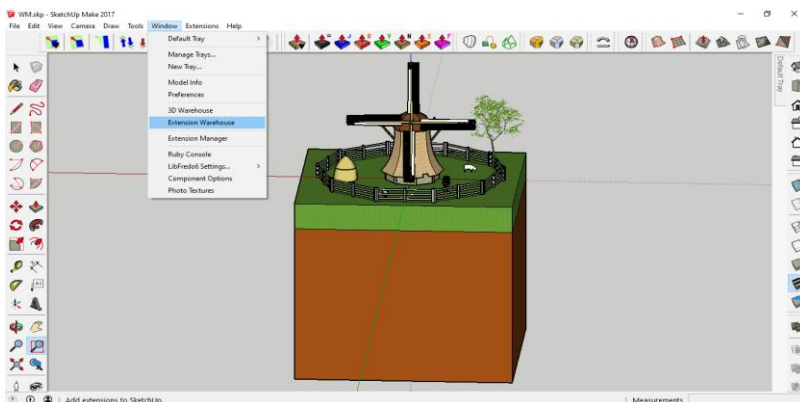


Figura 4.3. Accesarea opțiunii Extension Warehouse din bara de meniu

În figurile de mai jos sunt arătați pașii care trebuie urmați pentru crearea modelului *.obj.

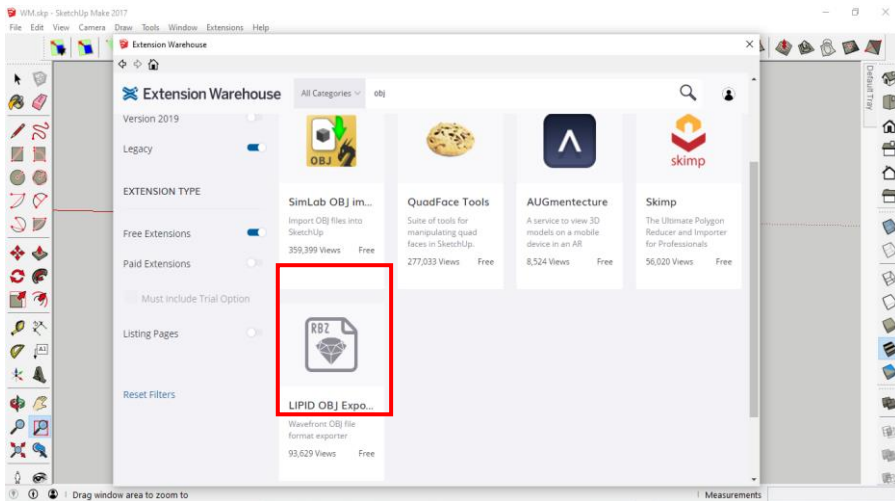


Figura 4.4. Fereastra de căutare și instalare a plugin-urilor din repoziitoriul online Extension Warehouse

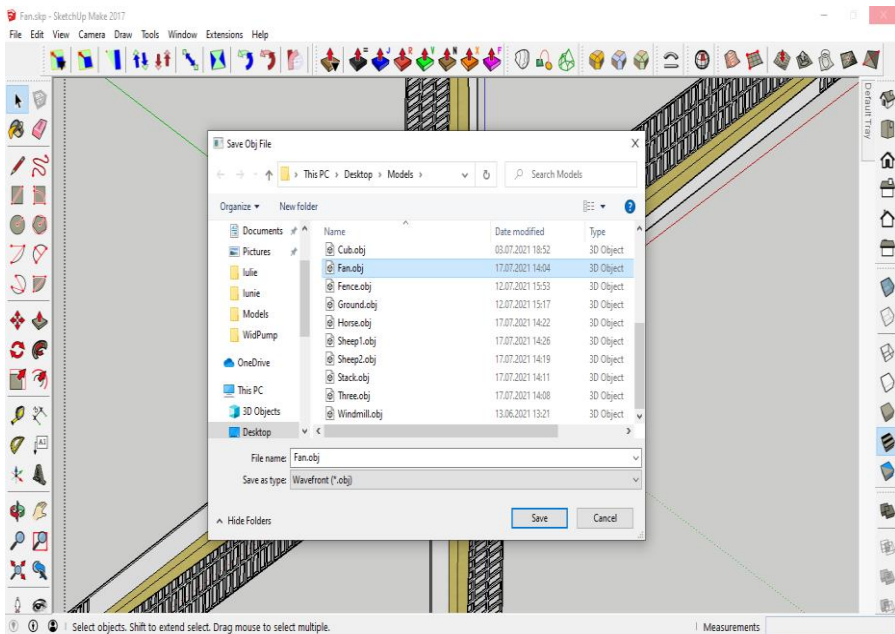


Figura 4.5. Fereastra de salvare în format .obj a modelelor 3D ale obiectelor din scenă

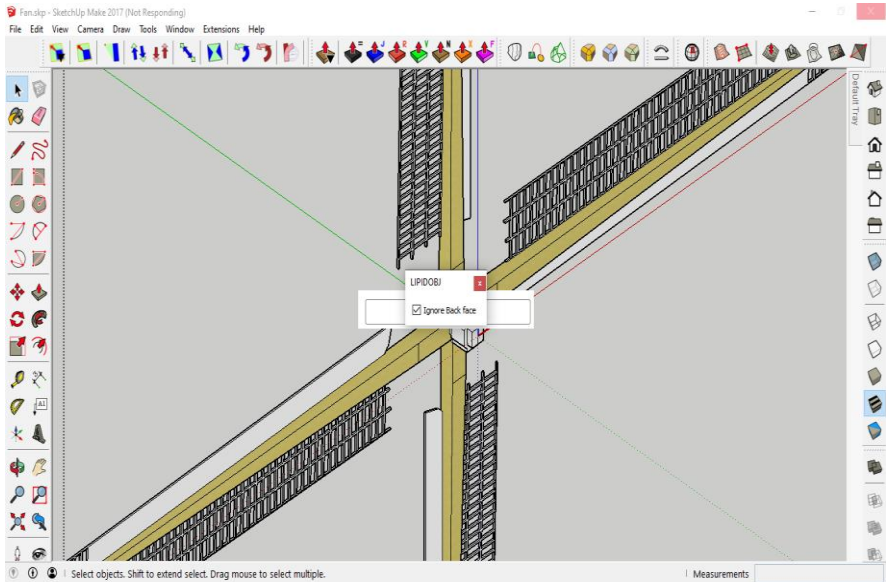


Figura 4.6. Opțiunea de ignorare a fețelor plugin-ului LIPIDOBJ la exportarea modelelor 3D în format .obj

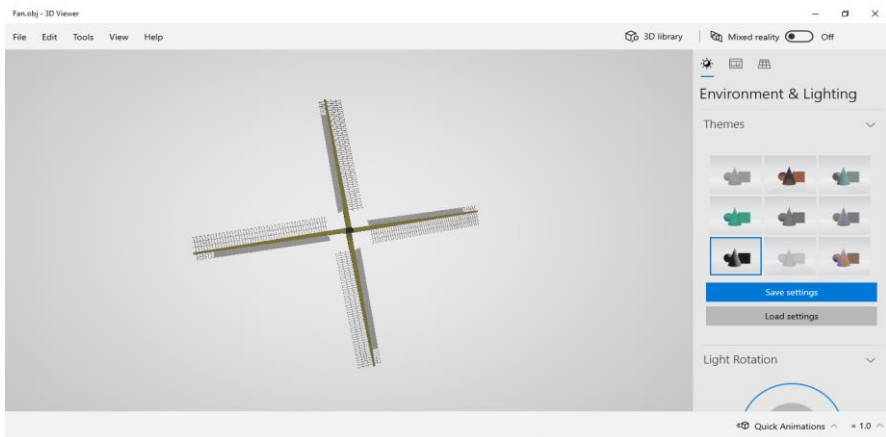


Figura 4.7. Vizualizarea conținutului fișierelor .OBJ cu ajutorul aplicației standard din cadrul sistemului de operare Windows 10–3D Viewer

Lucrarea de laborator nr. 4

Tema: CREAREA SCENEI STATICE 3D

Obiectivele lucrării:

1. Crearea unui model 3D static – se utilizează un editor grafic compatibil cu exportul în format .obj (de exemplu, Blender, SketchUp sau Warehouse).

2. Exportul corect al modelului – în formatul .obj și importarea acestuia într-un mediu de programare p5.js.

3. Adăugarea de primitive grafice 3D – în p5.js pentru completarea și personalizarea scenei 3D.

4. Explorarea funcționalităților p5.js pentru lucrul cu obiecte 3D prin adăugarea atributelor de culoare și lumină.

5. Aplicarea conceptelor de grafică pe calculator – poziționarea și structurarea scenei tridimensionale.

6. Crearea unei scene finale care să combine modelul 3D importat și elementele grafice adăugate direct în p5.js pentru un efect vizual coerent.

Numărul de ore necesare pentru realizare – 4 ore academice.

Scopul lucrării: obținerea cunoștințelor practice privind sinteza scenelor grafice 3D statice, utilizând funcțiile standard de reprezentare a modelelor 3D din biblioteca p5.js. Dezvoltarea competențelor de modelare și vizualizare a obiectelor 3D prin utilizarea editorilor grafici și integrarea modelelor în formatul *.obj în medii interactive bazate pe JavaScript.



Sarcina lucrării:






1. Elaborați un obiect cu extensia ***.obj** în care să creați o scenă 3D statică conform variantei indicate în tabelul 4.1. Pentru crearea scenei pot fi utilizate obiecte grafice 3D existente în repozitoriul 3D Warehouse.


2. Elaborați un program în care sintetizați o scenă 3D statică care ar conține cel puțin 5 obiecte, utilizând funcțiile standard de

reprezentare a modelelor 3D din biblioteca p5.js, în scena creată importată obiectul *.obj creat în punctul 1 al sarcinii.

Tabelul 4.1. Variante pentru realizarea lucrării de laborator

<i>Nr.</i>	<i>Obiect 3D</i>	<i>Model</i>
1	Carusel	
2	Far maritim	
3	Automobil blindat	
4	Moară de apă	

<i>Nr.</i>	<i>Obiect 3D</i>	<i>Model</i>
5	Avion elice cu	
6	Morișcă vânt de	
7	Turbină eoliană	
8	Elicopter	
9	Submarin	

<i>Nr.</i>	<i>Obiect 3D</i>	<i>Model</i>
10	Catapultă	

Exemplu:

```

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();
  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(); }

```

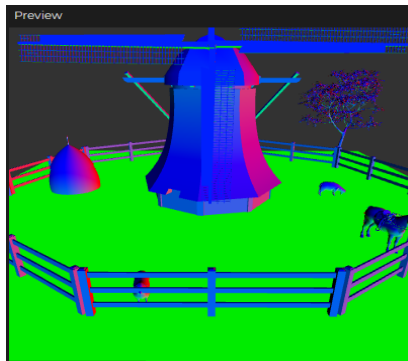


Figura 4.8. Rezultatul execuției programului

Criteria de evaluare

1. Corectitudinea modelării 3D în editorul grafic (20%) – dacă obiectul 3D realizat este conform cerințelor și bine definit, dacă exportarea în formatul .obj s-a realizat corect și fără pierderi de date.

2. Integrarea modelului 3D în p5.js (25%) – dacă modelul importat este funcțional în p5.js fără erori de afișare, dacă poziționarea modelului în scena 3D este corectă și adecvată.

3. Adăugarea și utilizarea primitivelor grafice în p5.js (25%) – dacă în scenă au fost adăugate primitive grafice 3D care

contribuie la compoziția generală, dacă primitivelor li s-au atribuit poziționări și dimensiuni adecvate pentru a crea o scenă coerentă.

4. Respectarea cerințelor și claritatea codului (10%) – dacă codul este clar, bine organizat și documentat, respectând cerințele tehnice specificate. Comentariile explicative sunt incluse unde este necesar pentru a clarifica funcționalitatea. Dacă scena este estetică și bine structurată, cu o compoziție echilibrată, sau au fost aplicate concepte de bază de grafică (iluminare, umbre, perspectivă) pentru o experiență vizuală plăcută.

5. Evaluarea cunoștințelor (10%) – explicații privind procesul de realizare a lucrării, ceea ce poate include descrierea funcțiilor principale și a logicii utilizate.

6. Respectarea cerințelor și termenului de prezentare finală a proiectului – (10%).

Întrebări de autoevaluare a cunoștințelor

1. Ce este un model 3D static și prin ce diferă de unul animat?
2. Explicați pașii necesari pentru a exporta un model 3D în format .obj și a-l importa într-un proiect p5.js.
3. Ce primitive grafice 3D în p5.js cunoașteți? Enumerați câteva exemple și aplicații posibile.
4. Cum se realizează rotirea, translația și scalarea unui obiect 3D în p5.js?
5. Care sunt avantajele utilizării p5.js pentru redarea scenelor 3D statice?
6. Cum puteți ajusta iluminarea în scena 3D p5.js pentru a obține un efect realist?
7. Ce funcții și metode ați folosit pentru a adăuga elemente grafice 3D în p5.js și cum au contribuit ele la aspectul scenei finale?

V. REALIZAREA SCENEI DINAMICE ÎN 3D

5.1. Introducere în realitatea augmentată

Realitatea augmentată (AR) este o tehnologie care îmbină elementele virtuale cu lumea reală, adăugând informații vizuale, sonore sau alte tipuri de date digitale la ceea ce utilizatorul vede în mediul înconjurător. Acest proces se realizează prin dispozitive precum smartphone-uri, tablete, ochelari de realitate augmentată sau alte echipamente ce pot proiecta sau suprapune obiecte digitale peste imagini reale.

Realitatea augmentată combină mai multe tehnologii pentru a crea o aplicație captivantă care constă din următoarele aspecte:

1. Detectarea și urmărirea – identifică poziția obiectelor din lumea reală pentru a putea proiecta elementele virtuale într-o locație precisă.

2. Senzori și camere – dispozitivele folosesc camere, accelerometre și giroscopuri pentru captarea informației despre mediul înconjurător.

3. Afișaj – datele augmentate sunt proiectate pe ecranul dispozitivului sau prin ochelari de RA, suprapunând obiecte virtuale peste realitatea fizică.

4. Software și algoritmi de recunoaștere – aplicațiile folosesc algoritmi pentru analiza și procesarea datelor capturate, identificând obiectele și poziționând corect elementele virtuale.

Deci, realitatea augmentată este o tehnologie care suprapune elemente digitale, cum ar fi obiectele grafice 2D sau 3D, sunete și alte efecte vizuale peste lumea reală. În această lucrare va fi explorată platforma Artivive pentru crearea unei scene augmentate. Un exemplu de aplicare a RA este arătat în figura 5.1.

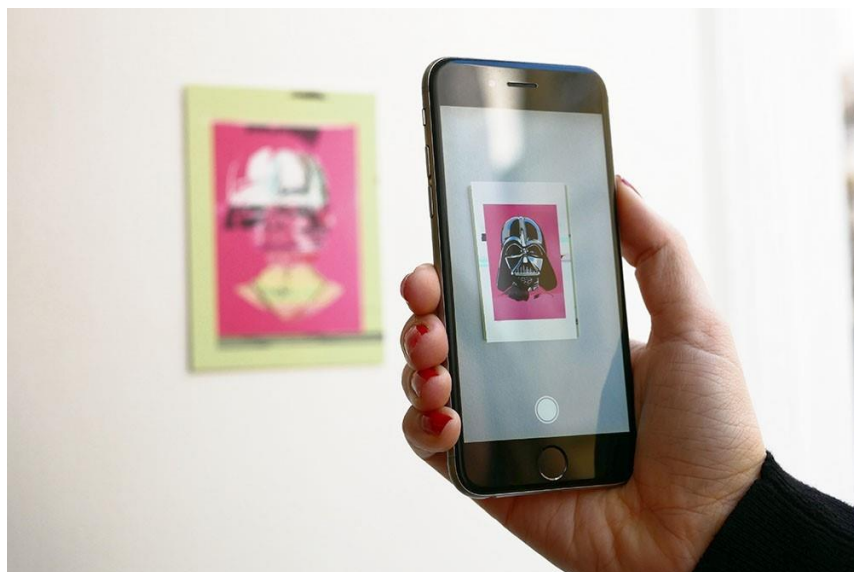


Figura 5.1. Realitatea augmentată

Pentru implementarea proiectelor în realitatea augmentată (AR), există mai multe platforme populare și versatile, fiecare oferind funcționalități adaptate diferitelor tipuri de proiecte. Platforme utilizate pentru crearea aplicațiilor RA sunt:

1. Unity + Vuforia - **Unity** este un motor de jocuri foarte popular care permite crearea aplicațiilor AR interactive. **Vuforia** este un SDK AR folosit cu Unity pentru recunoașterea imaginilor, urmărirea obiectelor și integrarea 3D. Permite o flexibilitate mare, fiind potrivit pentru aplicații complexe și oferă suport extins pentru animații și grafice 3D.

2. ARKit (Apple) este un framework dezvoltat de Apple pentru crearea experiențelor AR pe iOS. Se integrează excelent cu dispozitivele Apple, foarte eficient în recunoașterea feței, urmărirea mișcărilor și integrarea cu hardware-ul Apple.

3. ARCore (Google) este un SDK oferit de Google pentru crearea aplicațiilor AR pe dispozitivele Android. Oferă funcții de detectare a planului, estimarea luminozității și urmărirea mișcărilor; este gratuit și compatibil cu multe dispozitive Android.

4. Spark AR Studio (Meta) este folosit pentru a crea filtre AR pentru Instagram și Facebook. Accesibil pentru designeri și creatorii de conținut, permite crearea filtrelor personalizate și oferă o interfață intuitivă pentru implementare rapidă.

5. Lens Studio (Snapchat) este platforma Snapchat pentru crearea filtrelor AR, cunoscute ca „lenses.” Ușor de utilizat, cu instrumente vizuale pentru tracking facial, corp și obiecte și ideal pentru crearea de experiențe interactive rapide.

6. WebAR (8th Wall, AR.js) - 8th Wall și AR.js sunt platforme care permit crearea experiențelor AR direct în browser, fără a necesita aplicații. Utilizatorii pot accesa AR prin linkuri web fără instalarea aplicațiilor; este o soluție eficientă pentru marketing și proiecte AR bazate pe web.

7. Artivive este o platformă de AR specifică pentru artă și multimedia utilizată pentru suprapunerea animațiilor și a efectelor AR peste imaginile statice. Ideală pentru artiști și creatorii de conținut multimedia, permite o implementare simplă și rapidă a efectelor AR pe materiale fizice, cum ar fi postere sau ilustrații.

8. Adobe Aero este un instrument de la Adobe care permite crearea experiențelor AR fără cod. Ușor de utilizat pentru designeri, are o integrare bună cu alte aplicații Adobe, este potrivit pentru proiecte creative și de artă.

Diversitatea mare a platformelor de realitate augmentată reflectă flexibilitatea și potențialul imens al acestui domeniu, fiecare instrument oferind soluții adaptate diferitelor tipuri de proiecte și niveluri de experiență.

5.2. Platforma Artivive

Pentru a începe lucrul în platforma Artivive se vor urma pașii:

- **Descărcarea aplicația:** aplicația Artivive este disponibilă pe iOS și Android.

- **Creare cont:** accesați platforma Artivive și creați un cont de utilizator. Contul permite încărcarea proiectelor și vizualizarea acestora prin intermediul aplicației mobile. În figura 5.2 este arătată fereastra platformei Artivive.

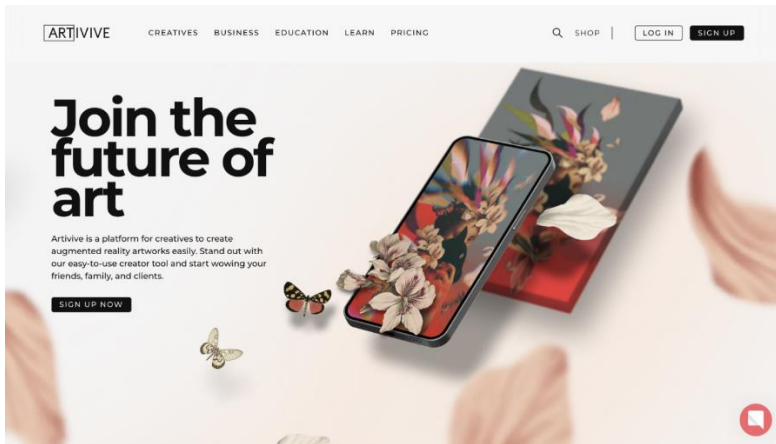


Figura 5.2. Platforma Artivive

- Accesați secțiunea **Tutoriale** de pe platformă pentru a înțelege cum să adăugați elemente grafice și să le suprapuneți pe imagini după cum este arătat în figura 5.3.

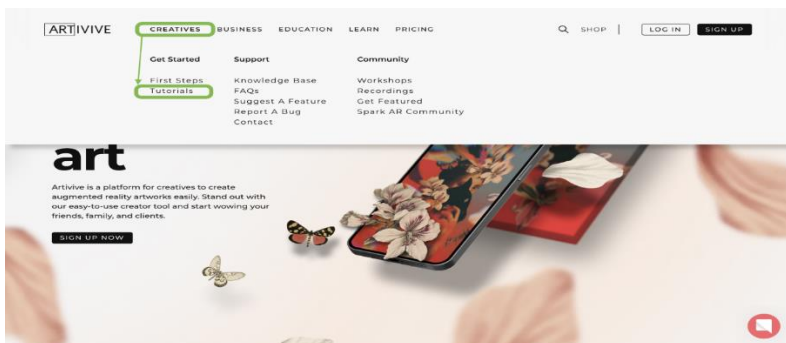


Figura 5.3. Secțiunea Tutorials

Urmați tutorialul disponibil în platformă pentru crearea primei scene simple augmentate, care poate fi scanată cu aplicația mobilă pentru a înțelege cum funcționează. Un exemplu de imagine până la scanare și imaginea după scanare este dat în figura 5.4.

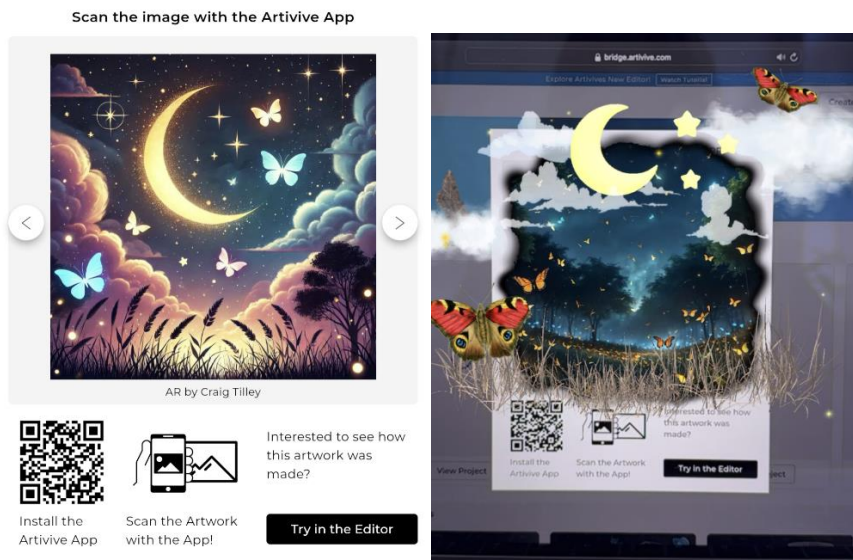


Figura 5.4. Pictură augmentată în starea inițială și după scanare

Interfața de lucru a platformei Artivive este simplă și intuitivă, fiind proiectată special pentru a facilita adăugarea elementelor AR peste imaginile statice.

Elementele principale ale interfeței:

1. Panoul de încărcare - secțiunea în care utilizatorii pot încărca imaginea de bază (trigger image) și conținutul AR (cum ar fi videoclipuri sau animații).

2. Previzualizarea AR - funcție de previzualizare care permite verificarea în timp real a modului în care arată efectele AR, utilizând aplicația Artivive pe telefon.

3. Setările proiect - include opțiuni pentru ajustarea dimensiunilor, a duratei efectelor AR, precum și alte setări de configurare.

4. Managementul fișierelor - utilizatorii pot gestiona și organiza fișierele pentru a facilita accesul și pentru a edita rapid conținutul adăugat.

Exemplul interfeței este dat în figura 5.5.

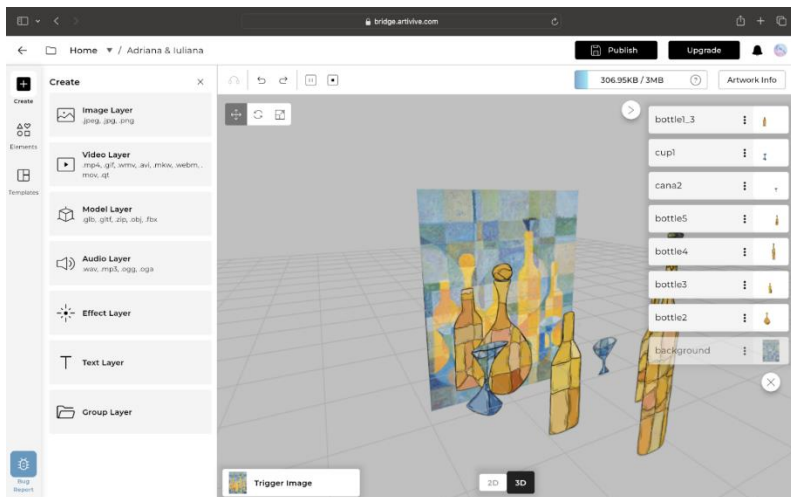


Figura 5.5. Interfața platformei Artive

Lucrarea de laborator nr. 5 Tema: REALIZAREA SCENEI DINAMICE 3D

Obiectivele lucrării:

1. Familiarizarea cu platforma Artive și studierea funcțiilor de bază ale acesteia pentru proiectele de realitate augmentată (AR).

2. Dezvoltarea scenei augmentate personalizate care să combine elemente grafice 2D și 3D.

3. Dezvoltarea abilităților practice de integrare a efectelor vizuale, a sunetului și animațiilor într-un proiect AR dinamic și creativ.

Numărul de ore necesare pentru realizare – 4 ore academice.

Scopul lucrării: crearea scenei augmentate originale, utilizând platforma Artive prin suprapunerea elementelor grafice,

animațiilor și a sunetului peste o imagine statică pentru a crea o compoziție dinamică și interactivă.

Sarcina lucrării: elaborați o scenă augmentată pe platforma Artivive (sau orice altă platformă pentru dezvoltarea proiectelor AR), alegând o temă care să te inspire. Scena trebuie să includă atât elemente grafice 2D și 3D, statice și dinamice, cât și sunet sau muzică pentru a completa atmosfera. Scena trebuie să conțină minim 5 elemente grafice.

Ghid pentru realizarea proiectului:

- *Definirea temei:* alegeți un subiect pentru compoziție care să permită exprimarea creativă.

- *Adaugați obiecte grafice:* folosiți elemente 2D și 3D. Puteți crea aceste elemente de sine stătător sau puteți să le descărcați din surse gratuite online (de ex.: Lumalabs.ai [Freepik](#), [Pexels](#), [Pixabay](#) etc.).

- *Integrați efectele și animațiile:* adăugați mișcarea și animații pentru a face scena dinamică. Platforma Artivive permite integrarea animațiilor video și elementelor interactive.

- *Adăugarea sunetului:* încorporați o melodie sau efecte sonore ce completează compoziția. Asigurați-vă că nu sunt încălcate drepturile de autor pentru sunetele alese.

- *Testarea și ajustarea:* verificați scena folosind aplicația Artivive pe telefon pentru a vedea cum arată și ajustați scena dacă nu sunteți mulțumit de rezultat.

Criterii de evaluare

1. Modelarea și animarea obiectelor 3D (25%) – conformitatea proiectului cu tema și respectarea cerinței minime de 5 elemente grafice, respectarea cerințelor tehnice legate de pregătirea fișierului pentru importul în Artivive.

2. Integrarea elementelor dinamice și a efectelor RA (25%)
– utilizarea creativă a platformei Artivive, inclusiv integrarea de elemente grafice și animații.

3. Estetica și coerența scenei dinamice (25%) – coerența estetică a compoziției prin combinarea armonioasă a elementelor vizuale și a sunetului, animațiile și obiectele sunt clare și bine realizate, atenție la detalii.

4. Capacitatea de testare și ajustare a scenei pentru a obține o experiență de utilizator fluentă și captivantă (20%).

5. Respectarea termenului de predare (5%).

Întrebări de verificare a cunoștințelor

1. Ce este realitatea augmentată și prin ce diferă de realitatea virtuală?

2. Ce rol joacă platforma Artivive în crearea proiectelor de realitate augmentată?

3. Cum poate fi adăugată o animație în proiectul AR folosind Artivive?

4. Care sunt principalele diferențe dintre un obiect grafic 2D și unul 3D în contextul AR?

5. Cum poate influența sunetul experiența utilizatorului într-un proiect de realitate augmentată?

6. De ce este importantă testarea și ajustarea scenei AR pe un dispozitiv mobil?

7. Cum pot fi utilizate platformele alternative la Artivive pentru a crea aplicații de AR?

VI. MODELAREA PROCESELOR 3D DINAMICE

Modelarea proceselor 3D dinamice, pe lângă scopuri pur științifice, poate avea și valoare aplicativă.

Biblioteca grafică p5.js poate fi utilizată pentru modelarea fizică a proceselor dinamice. Aceasta permite modelarea sistemelor mecanice (în cadrul legilor mecanicii teoretice). Cu ajutorul p5.js pot fi simulate mișcări de translație și rotație în trei planuri.

În exemplul prezentat în figura 6.1 este simulată ciocnirea a două sfere cu masele m_1 și m_2 , care până la ciocnire au vitezele deplasării liniare a centrelor notate cu v_1 și v_2 . După ciocnire, sferele se vor deplasa cu vitezele v'_1 și v'_2 . Sferele se rostogolesc pe o suprafață rigidă, adică execută o mișcare de rotație specificată de unghiuri: "omega₁" și "omega₂".

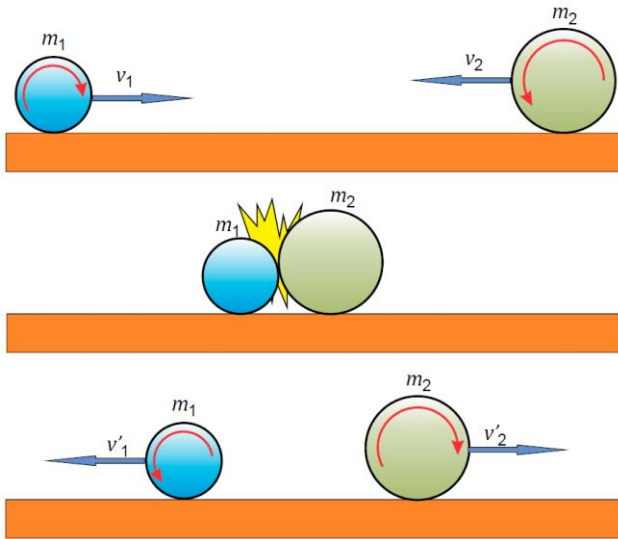


Figura 6.1. Ciocnirea a două sfere

Utilizând legea conservării energiei mecanice din formula (6.1):

$$\frac{m_1 v_1^2}{2} + \frac{m_2 v_2^2}{2} = \frac{m_1 v_1'^2}{2} + \frac{m_2 v_2'^2}{2} \quad (6.1)$$

și legea conservării impulsului din formula (6.2):

$$m_1 v_1 + m_2 v_2 = m_1 v'_1 + m_2 v'_2, \quad (6.2)$$

pot fi determinate relațiile dintre viteze până și după ciocnirea sferelor (formulele 6.3 și 6.4):

$$v'_1 = -v_1 + 2 \frac{m_1 v_1 + m_2 v_2}{m_1 + m_2}, \quad (6.3)$$

$$v'_2 = -v_2 + 2 \frac{m_1 v_1 + m_2 v_2}{m_1 + m_2}. \quad (6.4)$$

Unghiul de rotație poate fi exprimat cu ajutorul formulelor 6.5 și 6.6.

$$\omega_1 = \frac{r_1}{v_1}; \quad (6.5)$$

$$\omega_2 = \frac{r_2}{v_2}. \quad (6.6)$$

Modelul acestui experiment poate fi descris de următorul cod:

Listingul programului:

```
let x1=-150;
let v1=5;
let m1=50;
let omega1=v1/m1;
let a1=0;
let x2=150;
let v2=-15;
let m2=20;
let omega2=v2/m2;
let a2=0;
function setup()
{ createCanvas(500, 500, WEBGL);
  strokeWeight(0.2);
  frameRate(24); }
function draw()
{ background(200);
  orbitControl();
  pointLight(255, 255, 255, 0, 0, 1000);
```



```

push();
box(500,100,100);
pop();
push();
translate(x1, -100/2-m1, 0);
rotateZ(a1);
sphere(m1);
pop();
push();
translate(x2, -100/2-m2, 0);
rotateZ(a2);
sphere(m2);
pop();
if(x1+m1>x2-m2)
    { v1=2*(m1*v1+m2*v2)/(m1+m2)-v1;
      v2=2*(m2*v2+m1*v1)/(m1+m2)-v2;
      x1=x1+v1;
      x2=x2+v2;
      omega1=v1/m1;
      omega2=v2/m2;      }
      x1=x1+v1;
      x2=x2+v2;
      a1=a1+omega1;
      a2=a2+omega2;  }

```

Descrierea programului:

let – creează și denumește o variabilă nouă. O variabilă este un container pentru a memora a valoare.

Variabilele care sunt declarate cu **let** vor avea domeniul de aplicare bloc. Aceasta înseamnă că variabila există numai în blocul în care este creată.

În programul de mai sus, în calitate de variabile considerăm:

- x_1 și x_2 sunt pozițiile inițiale ale sferelor;
- v_1 și v_2 sunt vitezele liniare inițiale ale centrelor sferelor;
- masa bilelor m_1 și m_2 care exprimă și razele sferelor;

– $\omega_1 = v_1/m_1$ și $\omega_2 = v_2/m_2$ sunt unghiurile de rotație a sferelor;

– $a_1 = 0$ și $a_2 = 0$ sunt unghiurile inițiale de rotație a sferelor.

Corpul pe care se rostogolesc sferele este construit cu ajutorul funcției **box(500,100,100)**, fiind un obiect static.

Sferele sunt desenate cu ajutorul funcției **sphere()**, asupra lor sunt aplicate mișcarea de translație cu ajutorul funcției **translate()** și de rotație cu ajutorul funcției **rotateZ()**.

Modelele corpurilor sunt apelate fiecare în blocul său, fiind delimitate cu **push()**, **pop()**.

Ultima parte a programului include formulele de calcul pentru vitezele sferelor, pozițiilor acestora și al unghiului de rotație.

Rezultatul execuției programului este dat în figura 6.2.

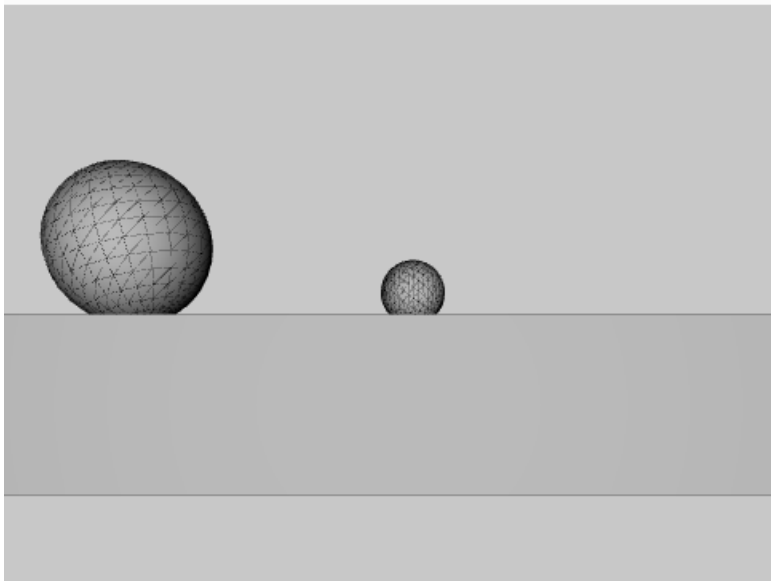


Figura 6.2. Modelul simulării ciocnirii a 2 sfere în p5.js

Pentru realizarea modelului se utilizează primitivele 3D din biblioteca p5.js.

6.1 Crearea primitivelor grafice 3D simple

Pentru crearea corpurilor în p5.js sunt utilizate primitivele grafice 3D simple cum ar fi:

- plane()
- box()
- sphere()
- cylinder()
- cone()
- elipsoid()
- torus()

Funcția plane(): desenează un plan cu o lățime și o înălțime dată

Sintaxa:

```
plane([width],[height], [detailX], [detailY]);
```

unde:

[width]: lățimea (opțional)

[height]: înălțimea (opțional)

detailX: numărul de triunghiuri pe axa x (opțional)

detailY: numărul de triunghiuri pe axa y (opțional)

Funcția box(): desenează un paralelepiped cu lățimea, înălțimea și adâncimea specificată

Sintaxa:

```
box([width],[Height],[depth],[detailX],  
[detailY]);
```

unde:

[width]: lățimea figurii (opțional)

[height]: înălțimea figurii (opțional)

[depth]: adâncimea figurii (opțional)

detailX: numărul de triunghiuri pe axa x (opțional)

detailY: numărul de triunghiuri pe axa y (opțional)

Funcția sphere(): Desenează o sferă cu raza specificată. DetailX și detailY determină numărul de segmente în dimensiunea x și dimensiunea y a sferei. Mai multe segmente fac sfera mai netedă.

Valoarea maximă recomandată este 24. Utilizarea unei valori mai mari de 24 poate face lucrul browserului mai lent.

Sintaxa:

```
sphere([radius], [detailX], [detailY]);
```

unde:

radius: raza cerdcului (opțional)

detailX: numărul de segmente pe axa x (opțional)

detailY: numărul de segmente pe axa y (opțional)

Funcția cylinder(): desenează un cilindru cu raza și înălțimea specificată. DetailX și detailY determină numărul de segmente pe axa x și axa y a cilindrului. Mai multe segmente fac ca cilindrul să pară mai neted. Valoarea maximă recomandată pentru detailX este 24. Utilizarea unei valori mai mari de 24 poate încetini browserul.

Sintaxa:

```
cylinder([radius], [height], [detailX],  
[detailY], [bottomCap], [topCap]);
```

unde:

radius: raza bazei cilindrului (opțional)

height: înălțimea cilindrului (opțional)

detailX: numărul de segmente pe axa x, implicit 24 (opțional)

detailY: numărul de segmente pe axa y, implicit 1 (opțional)

bottomCap Boolean: să deseneze sau nu partea de jos a cilindrului (opțional)

topCap Boolean: să deseneze sau nu partea de sus a cilindrului (opțional)

Funcția cone(): desenează un con cu rază și înălțimea dată. DetailX și detailY determină numărul de segmente pe axa x și axa y a conului. Mai multe segmente fac conul să pară mai neted. Valoarea maximă recomandată pentru detailX este 24. Utilizarea unei valori mai mari de 24 poate încetini browserul.

Sintaxa:

```
cone([radius], [height], [detailX], [detailY],  
[cap]);
```

unde:

radius: raza bazei conului (opțional)

height: înălțimea conului (opțional)

detailX: numărul de segmente pe axa x, implicit 24 (opțional)

detailY: numărul de segmente pe axa y, implicit 1 (opțional)

Cap Boolean: să deseneze sau nu partea baza conului (opțional)

Funcția ellipsoid(): desenează un elipsoid. DetailX și detailY determină numărul segmente pe axa x și axa y a figurii. Mai multe segmente fac elipsoidul mai neted. Evitați numărul de detalieri mai mare de 150, deoarece acesta poate bloca browserul.

Sintaxa:

```
ellipsoid([radiusx], [radiusy], [radiusz],  
[detailX], [detailY]);
```

unde:

radiusx: x-raza elipsoidului (opțional)

radiusy: y- raza elipsoidului (opțional)

radiusz: z- raza elipsoidului (opțional)

detailX: numărul de segmente pe axa x, implicit este 24. Evitați numărul segmentelor mai mare de 150 poate bloca browserul. (Opțional)

detailY: numărul de segmente pe axa y, implicit este 16. Mai mare de 150 poate bloca browserul. (Opțional)

Funcția torus(): desenează un inel. DetailX și detailY determină numărul de segmente pe axa x și axa y a torului. Valorile maxime implicite pentru detailX și detailY sunt 24 și, respectiv, 16. Setarea lor la valori relativ mici, cum ar fi 4 și 6, vă permite să creați noi forme, altele decât un tor.

Sintaxa:

```
torus([radius], [tubeRadius], [detailX],  
[detailY]);
```

Parametrii:

radius: raza externă a figurii (Opțional)

tubeRadius: raza internă a figurii (Opțional)

detailX: numărul de segmente pe axa x, implicit este 24 (opțional).

detailY: numărul de segmente pe axa y, implicit este 16 (opțional).

Lucrarea de laborator nr. 6

Tema: MODELAREA PROCESELOR ÎN 3D

Obiectivele lucrării:

1. Familiarizarea cu procesele fizice utilizate pentru crearea scenelor grafice 3D.

2. Utilizarea primitivelor grafice – obținerea cunoștințelor practice privind utilizarea primitivelor grafice 3D pentru crearea scenei.

3. Crearea scenelor grafice 3D dinamice – dezvoltarea abilităților de creare și compunere a scenelor grafice 3D dinamice prin combinarea eficientă a transformărilor geometrice.

4. Înțelegerea și aplicarea conceptelor de procese fizice – utilizarea noțiunilor de viteză, putere, gravitație și altele utilizate pentru dirijarea dinamicii vizuale în grafica 3D.


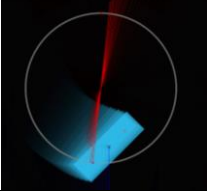
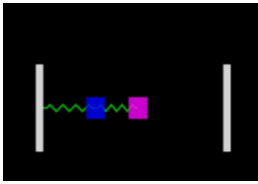
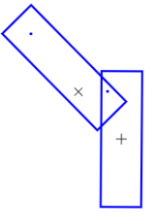
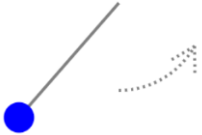
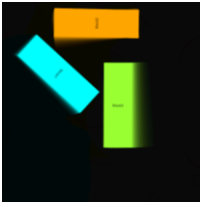
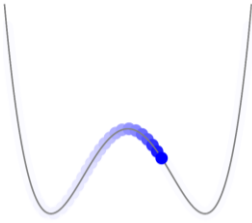
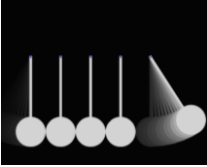
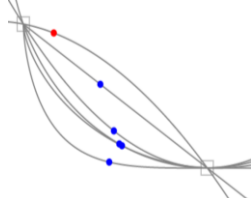
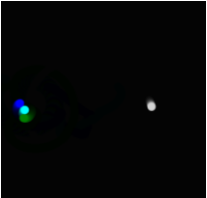
Numărul de ore necesare pentru realizare – 4 ore academice.

Scopul lucrării: obținerea cunoștințelor practice privind crearea scenelor 3D în modelarea proceselor 3D dinamice prin utilizarea funcțiilor standard de transformare geometrică, cum ar fi translația, scalarea și rotația din biblioteca p5.js.

Sarcina lucrării: elaborați un program care creează o scenă 3D în care să modelați un proces fizic, utilizând funcțiile standard de translație și rotație din biblioteca p5.js, conform variantei indicate în tabelul 6.1. Pentru crearea scenei pot fi utilizate obiecte grafice 3D existente în repozitoriul 3D.

Simularea și descrierea procesului fizic bidimensional din sarcină pot fi consultate pe pagina din [6].

Tabelul 6.1. Variante pentru realizarea lucrării de laborator

<i>Nr.</i>	<i>Model</i>	<i>Reprezentare</i>	<i>Nr.</i>	<i>Model</i>	<i>Reprezentare</i>
1	Arc de extensie singular		6	Pendul rigid cu mișcare pe cerc	
2	Arc de extensie dublu		7	Pendul dublu rigid	
3	Pendul haotic		8	Coliziunea blocurilor rigide	
4	Mișcarea sferei pe traiectorie de tip cocoașă		9	Pendulul lui Newton	
5	Mișcarea sferei pe traiectorii		10	Atracția corpurilor	

Criteria de evaluare

1. **Corectitudinea codului** (20%) – verificarea corectitudinii codului, fără erori de sintaxă și funcționare.

2. **Utilizarea primitivelor grafice 3D** (20%) – evaluarea utilizării corecte și varietatea primitivelor grafice în conformitate cu cerințele lucrării.

3. **Respectarea instrucțiunilor și cerințelor** (10%) – verificarea corectitudinii cerințelor sarcinii cum ar fi anumite forme sau dimensiuni specifice ale primitivelor grafice și dinamica lor.

4. **Optimizarea codului** (10%) – evaluarea eficienței codului în utilizarea resurselor și evitarea codului redundant.

5. **Interactivitatea** (10%) – răspunsuri la mouse sau tastatură, se va evalua dacă aceste funcții sunt implementate corect.

6. **Estetica vizuală** (10%) – analiza atractivității vizuale a compoziției, cum ar fi armonia culorilor, proporțiile și echilibrul grafic.

7. **Respectarea termenului de predare** (10%) – evaluarea punctajului în funcție de punctualitate, dacă lucrarea a fost predată în termenul stabilit.

8. **Evaluarea cunoștințelor** (10%) – explicații privind procesul de realizare a lucrării, ceea ce poate include descrierea funcțiilor principale și a logicii utilizate.

Întrebări de verificare a cunoștințelor

1. Enumerați primitivele grafice 3D simple.
2. Enumerați funcțiile utilizate pentru transformările grafice.
3. Cum poate fi realizată modificarea atributelor de afișare a primitivelor grafice 3D?
4. Numiți formate de rotație și funcțiile corespunzătoare.
5. Ce primitive grafice pot fi utilizate pentru a crea forme de bază într-o scenă grafică 2D?
6. Cum poate fi realizată combinarea transformărilor grafice?
7. Ce rol are funcția push() și pop() în gestionarea transformărilor geometrice?
8. Cum se activează modul 3D în p5.js?

VII. CREAREA MODELELOR PENTRU IMPRIMAREA 3D

Imprimarea 3D este procesul de formare a unui obiect tridimensional solid de orice formă, realizat printr-un proces aditiv în care straturile succesive de material sunt așezate în diferite forme. Imprimarea 3D diferă de metodele tradiționale de prelucrare, care se bazează pe îndepărtarea materialelor prin metode precum tăierea sau găurirea (procese substructive).

Imprimantele 3D permit designerilor să producă un prototip într-un timp restrâns. Prin urmare, prototipul poate fi testat și reproiectat rapid. Producția pieselor sau componentelor, formelor extrem de complexe prin metode clasice durează destul de mult, dar utilizarea acestei tehnologii de imprimare poate reduce semnificativ acest timp. Timpul economisit face posibilă testarea mai multor variante de componente pentru a dezvolta cât mai rapid soluția necesară.

Imprimarea 3D a unui obiect se face printr-un proces de adăugare controlată a materialului, strat cu strat, până când întregul obiect este creat exact așa cum a fost definit digital. Fiecare strat poate fi gândit ca o secțiune transversală orizontală a unui obiect sau mai precis o felie bidimensională, toate straturile fiind unite treptat pentru a forma forma finală a obiectului. Deci, imprimarea 3D este un proces destul de complex, dar, în general, procesul se rezumă la adăugarea noilor straturi peste cele existente pentru a forma o a treia dimensiune: înălțimea.

Toate imprimantele 3D moderne folosesc acest proces de adăugare strat cu strat și există mai multe tipuri de tehnologii, diferența dintre care constă în modul în care straturile sunt create și lipite între ele. Unii se bazează pe topirea sau creșterea gradului de ductilitate a materialului cu care sunt imprimate, alții se bazează pe diverse procese care presupun folosirea fasciculelor laser sau a luminii ultraviolete pe materialele susceptibile.

7.1. Tehnologiile de printare 3D

Industria imprimării 3D s-a dezvoltat mult în ultimii ani, crescând numărul modalităților de creare a unui obiect. Există mai multe moduri de a transforma consumabilele în produse finite. Unele metode topesc sau înmoaie materialul pentru a crea straturi, cum ar fi tehnologiile FDM și SLS, care sunt cele mai comune tehnologii în industria imprimării 3D.

FDM – Fused Deposition Modeling – Modelarea prin Extrudare Termoplastică

Tehnologia de prototipare rapidă FDM (Fused Deposition Modeling), tradusă ca modelare prin extrudare termoplastică (fuziunea materialelor), este cea mai utilizată tehnologie de fabricație aditivă datorită simplității și disponibilității sale. Este utilizată în modelarea, prototiparea, precum și în aplicații de producție. Alte denumiri folosite: MEM (Melting Extrusion Modeling), TPE (Thermoplastic Extrusion), FFF (Fused Filament Fabrication).

Folosind o aplicație software specială, modelul 3D dorit este inițial tăiat în secțiuni transversale numite straturi. Tehnologia de imprimare presupune trecerea unui filament de plastic printr-un extruder, care îl încălzește până la punctul de topire, apoi depunerea lui uniformă (prin extrudare) strat cu strat cu mare precizie pentru a imprima fizic un model 3D conform fișierului CAD.

Capul (extruderul) este încălzit pentru a topi filamentul de plastic, mișcându-se atât pe orizontală, cât și pe verticală sub controlul numeric dirijat direct de aplicația CAM a imprimantei. Pe măsură ce capul se mișcă, acesta aplică o bandă subțire de plastic extrudat, care se întărește imediat când este răcit, lipindu-se de stratul anterior și creând designul 3D dorit.

Pentru a preveni deformarea pieselor cauzate de răcirea bruscă a plasticului, unele modele profesionale de imprimare 3D includ o cameră de construcție închisă încălzită la o temperatură ridicată. Pentru geometriile complexe, tehnologia FDM necesită imprimarea cu material suport, care ulterior trebuie îndepărtat manual.

Aplicațiile tehnologiei FDM/MEM: aplicații ale truselor și hobby-urilor de imprimare 3D: design conceptual de piese simple cu precizie scăzută, imprimare hobby (1-2 părți/zi).

SLA – Stereolithography – Stereolitografia

Stereolitografia (SLA sau SL) este o tehnologie de prototipare rapidă utilizată larg în mediile industriale pentru a produce matricii, modele și chiar componente funcționale. Cunoscută și sub denumirea de fotopolimerizare sau fabricație optică, stereolitografia implică utilizarea unui fascicul laser cu lumină ultravioletă pentru a întări o rășină fotopolimer lichidă în rezervorul de imprimare al unei imprimante. Când este expusă la lumina laser ultravioletă, această rășină curabilă (sensibilă la lumina ultravioletă) se întărește în straturi succesive, obținând astfel un model 3D solid.

Modelul 3D dorit este inițial tăiat în secțiuni transversale, care sunt urmărite de un fascicul laser pe suprafața rășinii lichide. Expunerea la lumina laser ultravioletă întărește designul desenat pe rășina lichidă, rezultând un strat solid construit (imprimat 3D), care este adăugat stratului construit anterior.

Când construcția este finalizată, modelul 3D rezultat este scufundat într-o baie chimică separată pentru a îndepărta excesul de rășină, apoi este tratat într-un cuptor UV pentru întărirea finală.

Pentru a imprima geometrii complexe, stereolitografia necesită crearea unor structuri de susținere care să stabilizeze geometria obiectului. Aceste structuri sunt generate automat în timpul pregătirii 3D pe un computer folosind software-ul de imprimantă 3D. Când construcția este finalizată, suporturile vor trebui îndepărtate manual. Rășina rămasă în containerul de construcție poate fi reutilizată pentru imprimări ulterioare.

Aplicațiile tehnologiei SLA: piese și componente detaliate, modele gata pentru prezentări de marketing, testarea formei fizice, modele de scule rapide, produse rezistente la temperaturi ridicate, matricii master de turnare.

DLP – Digital-Light Processing – Expunerea digitală a luminii

Tehnologia de imprimare DLP (Digital Light Processing) este un proces de fabricație aditiv care utilizează lumina UV pentru a întări rășinile polimerice lichide. Dezvoltată de Texas Instruments, tehnologia DLP are ca element de bază un cip DMD (Digital Micromirror Device) – o serie de microoglinzi utilizate pentru modularea rapidă a luminii.

Mai întâi, modelul CAD 3D este convertit de software-ul imprimantei 3D în secțiuni transversale (slice) ale obiectului, apoi informațiile sunt trimise la imprimantă și la cipul DMD.

Pentru fiecare secțiune transversală a modelului CAD 3D, lumina UV emisă de proiector este modulată și proiectată prin cip pe suprafața rășinii din baia de lucru. Fiecare microoglinză individuală de pe cipul DMD proiectează pixeli dintr-o secțiune transversală a modelului 3D. Când este expusă la radiații UV, rășina lichidă fotoreactivă (sensibilă la lumina ultravioletă) se întărește în straturi succesive.

Aplicațiile tehnologiei DLP: prototipuri durabile pentru testarea funcțională, prototipuri și modele subțiri, precise (bijuterii, modele dentare, modele electronice), prototipuri cu geometrii complexe; producerea unor serii mici de modele în medicină (proteze auditive, restaurări dentare, implanturi medicale), prototipuri și modele în mass-media (animație, cinema etc.), modele pentru turnarea bijuteriilor, unelte și echipamente, piese și componente în industria auto și industria aerospațială.

SLS – Selective Laser Sintering – Sinterizare Laser Selectivă

Tehnologia de prototipare rapidă SLS (Selective Laser Sintering), care se traduce prin sinterizarea selectivă cu laser, a fost brevetată la sfârșitul anilor 1980 și este apropiată de SLA. Pe lângă denumirea SLS, denumirea generică LS (Laser Sintering) este, de asemenea, utilizată pe scară largă.

Tehnologia SLS presupune folosirea unui fascicul laser puternic (cum ar fi un laser CO₂) pentru a topi (sinteriza) pulberile în straturi succesive, obținându-se astfel modelul 3D dorit.

Modelul 3D dorit este inițial convertit în secțiuni transversale (slice) ale obiectului, apoi trimis la imprimantă. Pe baza informațiilor primite, un fascicul laser în mișcare topește selectiv (sinterizează) un strat de pulbere pe platforma de construcție din interiorul cuvei în funcție de fiecare secțiune transversală.

Când secțiunea transversală este completă, platforma pe care sunt construite modelele 3D este coborâtă în rezervor pentru a putea fi realizată următoarea secțiune transversală. Se aplică un nou strat de pulbere, care este apoi nivelat, după care procesul se repetă până când întregul model 3D este finalizat conform fișierului CAD.

În timpul imprimării, modelul 3D este încorporat permanent în pulberea de construcție, permițând imprimarea unor geometrii extrem de complexe fără material suport. Pulberea rămasă în rezervorul de constructor poate fi reutilizată pentru imprimări ulterioare.

Aplicațiile tehnologiei SLS/LS: piese rezistente la testele funcționale, testele la temperaturi ridicate, piese cu balamale și unități de montaj, producție la scară mică, modele de turnătorie.

SLM (DMLS) – Selective Laser Melting – Sinterizarea directă (topire) Laser a Metalelor

Tehnologia SLM (topirea selectivă cu laser) sau sinterizarea (topirea) cu laser a metalelor este o subsecțiune a tehnologiei SLS cu un proces similar de fabricație aditivă. Tehnologia se mai numește și DMLS (sinterizare directă cu laser a metalelor) sau LaserCusing.

Spre deosebire de sinterizarea selectivă cu laser, tehnologia SLM utilizează ca material de construcție pulberi metalice, care sunt topite și sudate împreună cu un laser de mare putere. Straturile subțiri de pulbere metalică automatizată se topesc și se solidifică succesiv la nivel microscopic în interiorul unei camere de construcție închise care conține un gaz inert (argon sau azot) în cantități strict controlate, la un anumit nivel de oxigen. Când este finalizată, partea 3D este

îndepărtată din camera de proiectare și tratată termic, apoi finisată pe baza aplicației.

Pentru uz industrial specializat, tehnologia de topire selectivă cu laser SLM poate fi folosită mai mult în domeniul prototipării rapide decât în domeniul imprimării 3D.

O tehnologie similară este EBM (electron beam melting), care utilizează un fascicul de electroni ca sursă de energie.

Aplicațiile SLM/DMLS: prototipuri robuste pentru testarea funcțională, piese cu geometrii organice, complexe și structuri cu pereți subțiri și goluri sau canale ascunse. Piese metalice complexe din materiale speciale, produse în serii mici. Forme hibride în care geometriile solide/parțiale/lattice pot fi combinate pentru a crea o singură entitate (de exemplu, implanturi ortopedice în care integrarea osoasă este îmbunătățită de geometria suprafeței).

3DP/3D inkjet printing – Printarea inkjet tridimensională

Tehnologia 3DP (Three-Dimensional Printing) se mai numește și imprimare 3D cu jet de cerneală sau imprimare 3D pe bază de gips (PP). Înainte de apariția LOM pe hârtie, 3DP era singura tehnologie care permitea imprimarea 3D color.

Imprimarea tridimensională 3DP implică utilizarea tehnologiei de imprimare prin injecție pentru a solidifica pulberea introdusă în camera de construcție (producție) a imprimantei prin lipirea particulelor împreună cu un material liant.

Inițial, modelul CAD 3D este convertit în secțiuni transversale (slice) ale obiectului, apoi direcționat spre imprimantă. În platforma de construcție se introduce un strat subțire de pulbere, după care se întinde, se distribuie și se comprimă uniform folosind o rolă specială. Capul de imprimare aplică apoi un jet de material liant, urmând structura proiectată (felie) a modelului 3D, rezultând un strat de obiect 3D format din pulbere de liant solidificat. Când stratul este complet, platforma de construcție este coborâtă exact la grosimea stratului, după care procesul de imprimare se reia.

Repetând operația, se construiesc straturi succesive, unul peste altul, până la realizarea piesei finite. Pe măsură ce procesul

progresează, piesa este scufundată în pulbere, ceea ce oferă un suport natural pentru geometrii mai complexe.

Partea finală se plasează într-o cuvă pentru a fi îndepărtată prin suflarea pulberii rămase în diferitele cavități și goluri. Culoarele pot fi adăugate, în timpul imprimării liantului, rezultând obiecte 3D colorate care sunt utile în multe aplicații.

În cazul pulberilor de amidon sau de gips, piesele imprimate 3D sunt de obicei impregnate cu un etanșant sau grund pentru a îmbunătăți duritatea și calitatea suprafeței. Pulberea rămasă în camera de construcție poate fi reutilizată pentru imprimări ulterioare.

Altă tehnologie de imprimare **3D for papier** combină imprimarea cu jet de cerneală cu tehnologia LOM. Straturile succesive de hârtie sunt tăiate în forme de secțiune transversală pentru a forma straturi ale modelului 3D și lipite împreună folosind un cap de imprimare care aplică un jet de adeziv. Tehnologia permite, imprimarea color a modelului 3D dorit folosind vopsele cu jet de cerneală obișnuite.

Aplicațiile tehnologiei 3DP/3D inkjet printing: imprimarea color este folosită în multe domenii în care aspectul vizual are maximă importanță: arhitectură, design conceptual, modele de marketing, vizualizare științifică, educație.

LOM – Laminated Object Manufacturing – Fabricarea Stratificată prin Laminare

Tehnologia LOM (Laminated Object Manufacturing) este o tehnologie mai puțin cunoscută, deși primul sistem de producție LOM a fost dezvoltat în 1991 de Helisys Inc.

Tehnologia LOM permite producerea strat cu strat a unui obiect 3D din straturi de hârtie sau plastic care sunt lipite unul peste altul și tăiate cu un cuțit sau cu laser. Materialul de imprimare utilizat poate fi furnizat fie în role (plastic), fie în coli (hârtie).

La început, modelul CAD 3D este convertit în secțiuni transversale (slice) ale obiectului, apoi trimis la imprimantă. Folosind o sursă laser sau un cuțit, imprimanta decupează straturile care vor alcătui partea 3D dintr-o foaie de material solid. Materialul rămas nefolosit după tăiere este tăiat fin cu un cuțit (sau sursă laser), astfel

încât să poată fi îndepărtat manual la sfârșitul procesului. Stratul finit este lipit de stratul anterior, folosind lipici aplicat pe fundul foii.

În timpul construcției, partea 3D este încadrată (învelită) într-un material structural, permițând tipărirea formelor geometrice complexe fără material suport. La sfârșitul procesului, partea 3D este înfășurată în exces de material, care va fi îndepărtat manual. Materialul rămas este aruncat și nu poate fi folosit pentru imprimări ulterioare.

O nouă tehnologie numită imprimare 3D pe hârtie combină imprimarea cu jet de cerneală cu tehnologia LOM. Secțiunile transversale ale hârtiei sunt mai întâi imprimate color, folosind tehnologia convențională cu jet de cerneală, apoi tăiate în straturi, rezultând un model 3D cu rezoluție completă a culorilor.

Aplicațiile tehnologiei LOM: experimentează forma fizică, modelul 3D volumetric (toate costurile de producție necesare pentru microfon), cu detalii maxime. Culoarea zonei de imprimare are multe aplicații, inclusiv aspecte vizuale importante: arhitectură, design conceptual, modele de marketing, randare vizuală, educație. Consiliul de servicii cu plată pentru tipar.

PJP – PolyJet Printing – Printare PolyJet cu Fotopolimeri

Tehnologia de imprimare 3D PJP (PolyJet Printing), cunoscută și sub denumirea de imprimare cu jet de cerneală fotopolimer sau imprimare cu jet multiplu (MJP), este o altă tehnologie de fabricație aditivă oarecum similară cu stereolitografia (SLA) prin faptul că folosește toată fotosolidificarea unui lichid fotopolimer. Cu toate acestea, tehnologia PolyJet este similară cu tehnologia convențională cu jet de cerneală. Spre deosebire de imprimantele de birou care pulverizează un flux de cerneală, imprimantele 3D PolyJet produc un flux de fotopolimeri lichizi care sunt ulterior întăriți prin expunerea la radiații UV.

Modelul CAD 3D este inițial convertit în secțiuni transversale (slice) ale obiectului, apoi direcționat spre imprimantă. Capul de imprimare pulverizează un flux de fotopolimeri lichizi cu care proiectează o secțiune de transfer foarte subțire pe platforma de construcție. Această secțiune este apoi consolidată cu ajutorul luminii

UV și procesul se repetă strat cu strat pentru a crea modelul 3D final. Modelele complete pot fi procesate și utilizate imediat, fără pași suplimentari de post-procesare.

Pentru geometrii complexe sau cantilever, imprimanta folosește un material suport asemănător gelului pentru a susține geometria. Acesta poate fi îndepărtat ulterior manual cu un jet de apă.

Imprimantele pot avea două sau mai multe capuri de imprimare, unul pentru fotopolimerul de construcție și celălalt pentru materialul solubil (gel). Folosind mai multe capuri, tehnologia PolyJet permite imprimarea a două materiale diferite în cadrul aceluiași proces de construcție. Astfel, prototipurile tipărite pot fi produse din diverse materiale cu proprietăți fizice diferite.

Aplicațiile tehnologiei PolyJet: piese și ansambluri robuste pentru testare funcțională, design conceptual, modele de prezentare și marketing, piese pentru o varietate de aplicații, producție în volum foarte redus, matricii de turnătorie, prototiparea rapidă a pieselor mici și a sculelor cu caracteristici complexe, matrițe master pentru turnarea uretanului.

Cea mai comună metodă este FDM, care a devenit populară datorită costului scăzut al imprimantelor/consumabilelor. Această metodă folosește filament PLA/ABS ca materie primă.

7.2. Formatul fișierelor STL

Procesul de imprimare 3D este strâns legat de acronimul STL, ce se referă la tipul formatului fișierilor (cu extensia *.stl), fiind foarte important, deoarece modelele 3D nu pot fi imprimate așa cum sunt și necesită pași intermediari.

Pentru ca un model să fie imprimat 3D, acesta trebuie să fie exportat în format STL, apoi trecut printr-un slicer, care îl „taie” pentru a crea un cod G pe care imprimanta 3D îl poate înțelege astfel, încât să poată fi create straturi până când piesa este finalizată.

Pentru imprimantele convenționale există un program, cum ar fi un cititor PDF sau un procesor de text etc., cu funcție de imprimare care, atunci când se accesează, trimite documentul în coada de imprimare pentru a putea fi tipărit. Cu toate acestea, imprimantele 3D

sunt complicate, deoarece necesită trei categorii de software pentru a funcționa.

Software utilizate pentru modelarea 3D: acestea sunt instrumente de modelare sau CAD în care se creează modelul ce urmează a fi imprimat.

Propunem mai jos câteva exemple:

- Blender
- Autodesk AutoCAD
- Autodesk Fusion 360
- Slash 3D
- Sketchup
- SolidWorks
- maya
- 3DS Max

Slicers este un tip de software care ia un fișier proiectat de unul dintre programele anterioare și îl feliază, adică îl taie în straturi. Deci, poate fi înțeles de imprimanta 3D, care construiește strat cu strat și îl convertește în codul G (un limbaj comun printre majoritatea producătorilor de imprimante 3D). Aceste fișiere includ și date suplimentare, cum ar fi viteza de imprimare, temperatura, înălțimea stratului, dacă există extrudare multiplă etc. În esență, este un instrument CAM care generează toate instrucțiunile pentru ca imprimanta să realizeze modelul.

Propunem mai jos câteva exemple:

- Ultimaker Cure
- Repeater
- Simplify3D
- slic3r
- KISSlicer
- ideaMaker
- Octo Print
- 3DPrinterOS

Gază de imprimantă sau software gazdă: imprimarea 3D se referă la un program a cărui funcție este de a primi un fișier GCode de la slicer și de a livra codul imprimantei printr-un port USB sau

printr-o rețea. Deci, imprimanta poate interpreta această „secvență” de comenzi GCode cu coordonatele X(0.00), Y(0.00) și Z(0.00) unde capul ar trebui să fie mutat pentru a crea obiectul și parametrii necesari. În multe cazuri, software-ul gazdă este integrat în slicer-ul propriu-zis, astfel încât acestea formează un singur program.

7.3. Slicer 3D

Slicer-ul, este un software ce feliază un model 3D, conceput pentru a produce straturile, formele și dimensiunile necesare pentru ca o imprimantă 3D să știe cum să-l creeze. În orice caz, procesul de tăiere în imprimarea 3D este destul de interesant, fiind o parte fundamentală a procesului. Acesta este un proces de tăiere pas cu pas care diferă ușor în funcție de tehnologia de imprimare 3D utilizată.

Astfel, se pot distinge:

- Tăierea FDM: aceasta necesită un control precis al mai multor axe (X/Y), deoarece mișcă capul de-a lungul a două axe și necesită multă mișcare a capului de imprimare pentru a crea un obiect 3D. Acesta include, de asemenea, parametri precum temperatura duzei și răcirea. Dacă slicerul generează GCode, algoritmi interni ai controlerului imprimantei vor fi responsabili pentru executarea comenzilor necesare.

- Tăierea SLA: în acest caz, comenzile trebuie să includă, de asemenea, timpul de expunere și viteza pasului. În locul așezării straturilor prin extrudare, trebuie să fie generat un fascicul de lumină pe diferite părți ale rășinii pentru a se întări și a crea straturi, ridicând simultan obiectul, astfel încât să poată fi creat un alt strat nou. Această tehnică necesită mai puțină mișcare decât FDM, deoarece doar oglinda reflectorizantă este controlată pentru a ghida laserul. De asemenea, este important de reținut că aceste tipuri de imprimante de obicei nu folosesc GCode, ci au propriile lor coduri (deci, au nevoie de propriul software de tăiere). Cu toate acestea, există câteva soluții SLA universale, cum ar fi ChiTuBox și FormWare care sunt compatibile cu multe imprimante 3D de acest tip.

- Tăierea DLP și MSLA: este similară cu SLA, dar cu diferența că singura mișcare necesară va fi mișcarea plăcii de lucru care se va

deplasa de-a lungul axei Z în timpul procesului. Alte informații vor fi orientate către panoul de afișare sau ecran.

- Altele: pentru altele, cum ar fi SLS, SLM, EBM etc., pot exista diferențe notabile în procesele de imprimare. Rețineți, în aceste trei cazuri menționate se adaugă o altă variabilă, cum ar fi injectarea de liant și este necesar un proces de feliere mai complex. Și la aceasta trebuie adăugat modelul de imprimantă SLS, nu funcționează la fel ca imprimanta SLS a unui concurent, de aceea este necesar un software special de tăiere (de obicei, programe proprii furnizate de producător).

7.4. Formatul STL

Formatul de fișier STL este un fișier cu toate cele necesare unui driver de imprimantă 3D, adică astfel încât hardware-ul imprimantei să poată imprima forma dorită. Cu alte cuvinte, permite codificarea geometriei suprafeței unui obiect 3D. A fost creat de Chuck Hull de la 3D Systems în anii 80.

Codificarea geometrică poate fi efectuată folosind teselația, inserând forme geometrice astfel, încât să nu existe suprapuneri sau goluri, adică, ca o placare. De exemplu, formele pot fi compuse folosind triunghiuri, așa cum este cazul redării GPU. O plasă fină formată din triunghiuri va forma întreaga suprafață a modelului 3D cu numărul de triunghiuri și coordonatele celor 3 puncte ale acestora.

STL binar/ASCII STL

Există o diferență între STL în format binar și STL în format ASCII și două moduri de a stoca și a prezenta informații despre aceste plăci și alți parametri.

Exemplu de format ASCII:

```
solid <nume>
facet normal nx ny nz
outer loop
vertex v1x v1y v1z
vertex v2x v2y v2z
vertex v3x v3y v3z
```

```
endloop
endfacet
endsolid <nume>
```

unde „vertex” sunt punctele necesare cu coordonatele XYZ respective.

Pentru a termina acest punct, trebuie știut că există o controversă cu privire la utilizarea sau nu a acestui tip de fișier.

7.5. Deschiderea fișierului STL

Una dintre metodele de a deschide un fișier STL este prin intermediul unor vizualizatoare online sau cu ajutorul software-ului instalat pe computer.

Online:

- [STL ViewSTL](#)
- [Autodesk Viewer](#)
- **Ferestre din:** [Microsoft 3D Viewer](#)
- **GNU/Linux:** [gmsk](#)
- **MACOS:** [Previzualizare sau Plăcut3D](#)
- **iOS / iPadOS:** [STL SimpleViewer](#)
- **Android:** [Vizualizator STL rapid](#)

7.6. Crearea fișierului STL

Pentru crearea fișierului STL sunt disponibile o diversitate mare de software pentru toate platformele și opțiuni online precum:

- **online:** [TinkerCAD](#), [Sketchup](#), [OnShape](#)
- **ferestre din:** [FreeCAD](#), [blender](#), [MeshLab](#)
- **GNU/Linux:** [FreeCAD](#), [blender](#), [MeshLab](#)
- **MACOS:** [FreeCAD](#), [blender](#), [MeshLab](#)
- **Sistem de operare iOS/iPad:**
- **Android:**

Există câteva aplicații de editare și modelare 3D pentru dispozitivele mobile, cum ar fi AutoCAD Mobile, Morphi, OnShape,

Prisma3D, Putty, Sculptura, Shapr3D etc., deși nu pot funcționa cu STL.

7.7. Alternativa fișierului STL

Recent au apărut câteva formate alternative pentru modelele de imprimare 3D. Aceste formate sunt, de asemenea, foarte importante și includ:

Unele dintre ele sunt *.gcode, *.mpt, *.mpf, *.nc etc.

- PLY (format de fișier poligon): aceste fișiere au extensia *.ply și sunt formate pentru poligoane sau triunghiuri. Ele au fost concepute pentru a stoca date 3D de la scanere 3D. Este o descriere geometrică simplă a unui obiect, precum și alte proprietăți, cum ar fi culoarea, transparența, normalele suprafeței, coordonatele texturii etc. Pentru STL există o versiune ASCII și o versiune binară.

- OBJ: fișierele cu extensia *.obj sunt, fișiere de definire a geometriei. Au fost dezvoltate de Wavefront Technologies pentru software numit Advanced Visualizer. Acum este open source și adoptat de multe programe de grafică 3D. De asemenea, stochează informații geometrice simple despre obiect, cum ar fi poziția fiecărui vârf, textura, normalul etc. Prin declararea vârfurilor în sens invers acelor ceasornicului nu trebuie declarate în mod explicit fețele normale. De asemenea, coordonatele în acest format nu au unități de măsură, dar pot conține informații despre scară.

- 3MF (3D Manufacturing Format): acest format este stocat în fișiere .3mf, un standard open source dezvoltat de consorțiul 3MF. Formatul de date geometrice pentru fabricarea aditivă se bazează pe XML. Poate include informații despre materiale, culoare etc.

- VRML (Virtual Reality Modeling Language): a fost creat de consorțiul Web3D. Aceste fișiere sunt într-un format al cărui scop este de a reprezenta scene sau obiecte 3D interactive, precum și culori de suprafață etc. Acestea stau la baza X3D (grafică 3D extensibilă).

- AMF (Additive Manufacturing Format): formatul fișierului (.amf) este un standard open source privind descrierea obiectelor pentru procesele de fabricație aditivă pentru imprimarea 3D. Este bazat pe XML și compatibil cu orice software de proiectare CAD. A

apărut ca succesori al STL, dar cu îmbunătățiri, cum ar fi includerea suportului nativ pentru culori, materiale, modele și constelații.

- WRL: extensie VRML.

7.8. Limbajul GCode

Limbajul de programare GCode este o parte-cheie a procesului de imprimare 3D, trecând de la designul STL la **un G-Code care este un fișier cu instrucțiuni și parametri de control ai imprimantei 3D**, o conversie care va fi efectuată automat de software-ul slicer.

Acest cod are comenzi care indică imprimantei cum și unde să extrudă materialul pentru a obține piesa de tipul:

- **G**: aceste coduri sunt înțelese universal de toate imprimantele care folosesc coduri G.
- **M**: acestea sunt coduri specifice pentru anumite serii de imprimante 3D.
- **Alte**: există și alte coduri native ale altor mașini, precum funcțiile F, T, H etc.

Fișierul GCode conține o serie de **linii de cod** care nu sunt altceva decât coordonate și alți parametri pentru a indica imprimantei 3D ce trebuie îndeplinit:

- **X, Y și Z**: sunt coordonatele celor trei axe de imprimare, adică ceea ce trebuie să deplaseze extruderul într-o direcție sau alta, coordonatele originii fiind 0,0,0.
- **F**: indica viteza cu care se deplasează capul de imprimare indicată în mm/min.
- **E**: se referă la lungimea extrudării în milimetri.
- **;**: tot textul care este precedat de ; este un comentariu și imprimanta îl ignoră.
- **G28**: se execută la început astfel, încât capul să se miște la oprirea. Dacă nu sunt specificate axele, imprimanta le va muta pe toate 3, dar dacă este specificată una anume, o va aplica doar aceleia.
- **G1**: una dintre cele mai populare comenzi G, deoarece este cea care impune imprimantei 3D să depună material în timp ce se deplasează liniar la coordonatele marcate (X,Y). De exemplu, G1 X1.0 Y3.5 F7200 indică depunerea materialului de-a lungul zonei

marcate de coordonatele 1.0 și 3.5 cu o viteză de 7200 mm/min, adică 120 mm/s.

- **G0:** acționează la fel ca G1, dar fără a extruda materialul, adică mișcă capul fără a depune material pentru acele mișcări sau zone în care nu trebuie depus nimic.

- **G92:** impune imprimanta să seteze poziția curentă a axelor sale, ceea ce este util când se dorește schimbarea locației axelor. Este utilizat chiar la începutul fiecărui strat sau în retragere.

- **M104:** comanda de încălzire a extruderului. Se folosește la început. De exemplu, *M104 S180 T0* ar indica faptul că extruderul T0 trebuie să fie încălzit (dacă există o duză dublă ar fi T0 și T1), în timp ce S determină temperatura, în acest caz 180°C.

- **M109:** similar cu cel de mai sus, dar indică faptul că imprimarea ar trebui să aștepte până când extruderul ajunge la temperatură, înainte de a continua cu orice alte comenzi.

- **M140 și M190:** asemănătoare celor două anterioare, dar nu au parametrul T, deoarece în acest caz se referă la temperatura patului.

Desigur, GCode funcționează **pentru imprimante de tip FDM**, deoarece cele din rășină vor avea nevoie de alți parametri.

Conversii STL

Cele mai utilizabile conversii:

- **Convertirea fișierelor Sketchup în STL** – poate fi realizată cu Sketchup într-un mod ușor, deoarece are atât funcții de import, cât și de export. În acest caz, trebuie efectuată exportarea urmând pașii: când fișierul Sketchup este deschis: Fișier > Export > Model 3D > se alege unde se salvează STL > Salvare ca fișier STereolithography (.stl) > Export.

• **Convertirea fișierelor STL în GCode** – poate fi convertit cu software de tăiere, deoarece este unul dintre obiectivele sale.

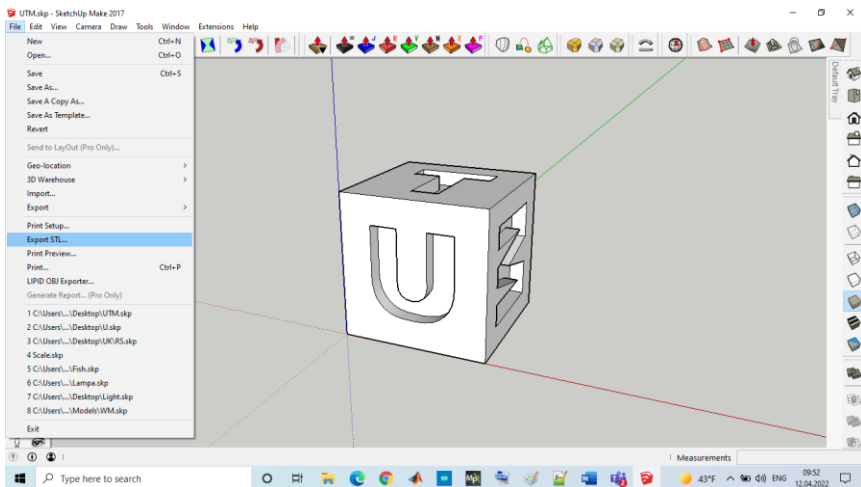


Figura 7.1. Exportarea fișierului în formatul STL

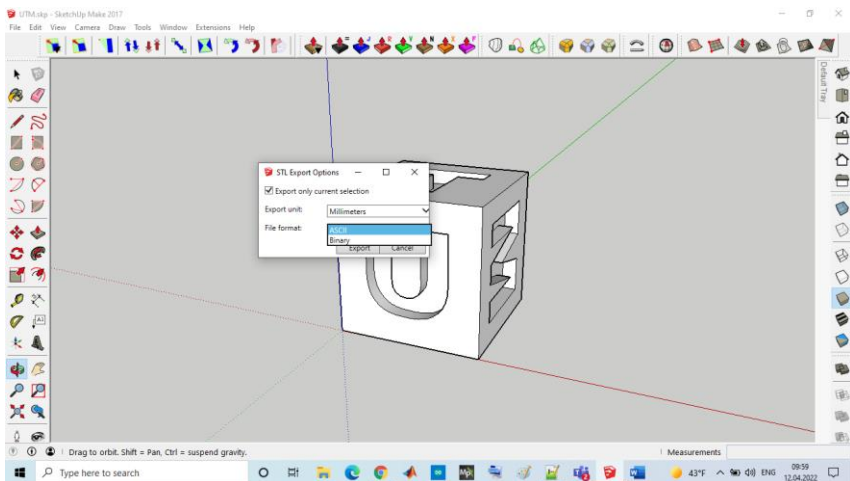


Figura 7.2. Cura

Adăugarea modelului 3D în Cura

Pentru importarea unui model al obiectului 3D în Cura se face click pe iconița mapei în partea stângă sus ori se selectează opțiunea: *File > Open File(s)* din meniul principal. Se selectează un fișier STL, OBJ sau 3MF de pe calculator. Acest lucru este arătat în figura 7.3.

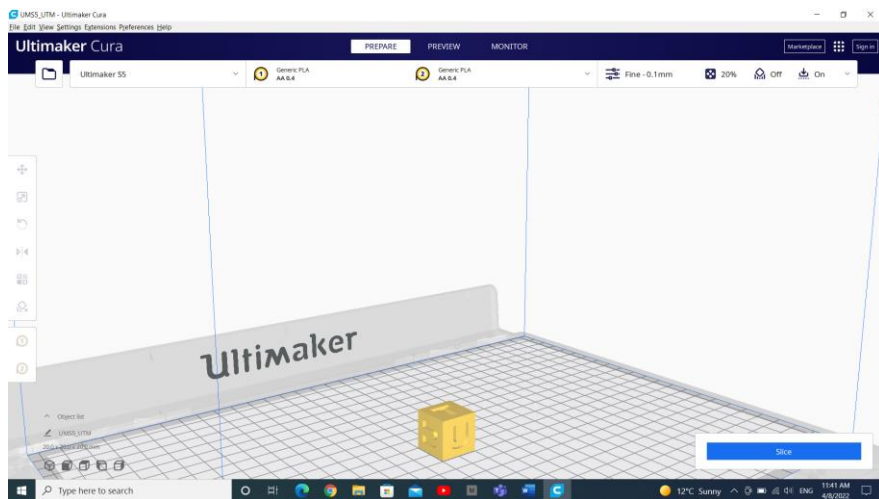


Figura 7.3. Importarea modelului 3D în Cura

Vizualizarea modelului în Cura

În softul Cura există trei căi de bază pentru vizualizarea modelului. Fiecare este folositor pentru diferite situații, în special când apare o problemă a obiectului de printat.

Solid – vizualizarea standard permite înțelegerea modului în care va arăta modelul printat. Vizualizarea mărimii și formei în dependență de platforma de imprimare. Dacă modelul arată bine și au fost folosite comenzile rapide doar pentru a naviga în jurul modelului.

X-Ray – se afla în meniul Preview; această caracteristică este indicată când printarea dă greș și permite rapid vizualizarea părților din interiorul structurii imprimării. Cel mai util este când imprimarea

constă dintr-o varietate de colțuri sau colțuri ce se intersectează cu alte colțuri. Comanda X-Ray în Cura permite vizualizarea aspectelor care trebuie refăcute.

Layers – în meniul Preview, dacă imprimarea dă greși într-un punct anumit sau a fost realizat ceva sofisticat și se dorește vizualizarea, dacă o parte a modelului de imprimare este bună poate fi accesată subcomanda Layer view. O soluție exactă este folosirea butoanelor arrow "sageți". O alternativă este un slide pentru vizualizarea rapidă printre toate straturile din care se constituie imprimarea utilizatorului. În Cura există abilitatea de a alege straturile cu ajutorul punctelor când este necesară schimbarea setarilor în G-code, pentru mărirea vitezei ventilatorului, înălțimea stratului sau debitul.

Control Infill Pattern - afectează rezistența modelului, consumul materialului și timpul printării.

În mod implicit, Cura slicer imprimă o umplutură în formă de grilă, imprimând într-o direcție diagonală pe strat. Acest lucru oferă o rezistență rezonabilă fără a consuma prea mult material. Este, de asemenea, unul dintre cele mai rapide modele în ceea ce privește timpul de imprimare. Modelul de umplere standard al Cura este indicat pentru cele mai comune aplicații.

În unele aplicații speciale, modelul implicit ar putea să nu fie cel mai bun. În astfel de cazuri, Cura oferă o gamă largă de modele de umplere la alegere.

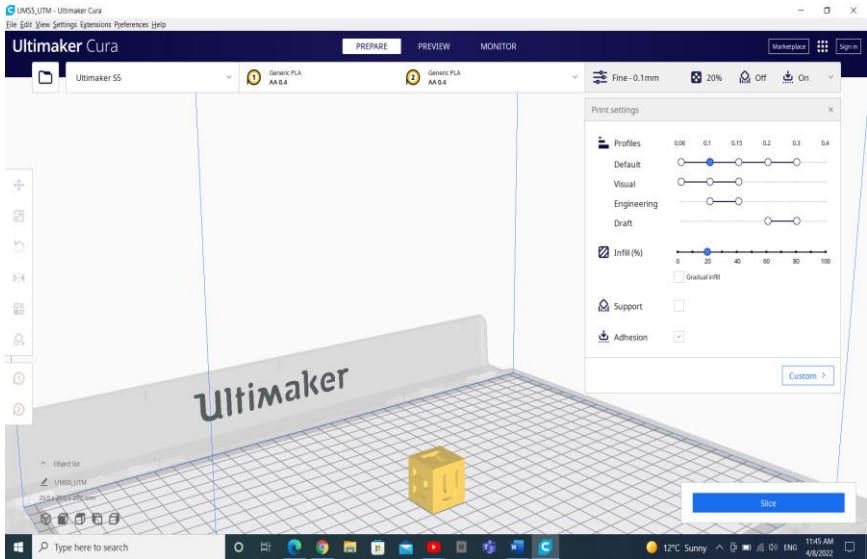


Figura 7.4. Setarea proprietăților modelului 3D și a slice-ului

Modele de umplere Cura3D. Există mai multe modele de umplere disponibile decât cele reprezentate în figura 7.5.

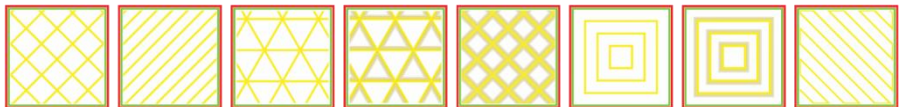


Figura 7.5. Exemple de umplerea a modelului

În figura 7.6 poate fi observat slice-ngul unui model 3D în Cura.

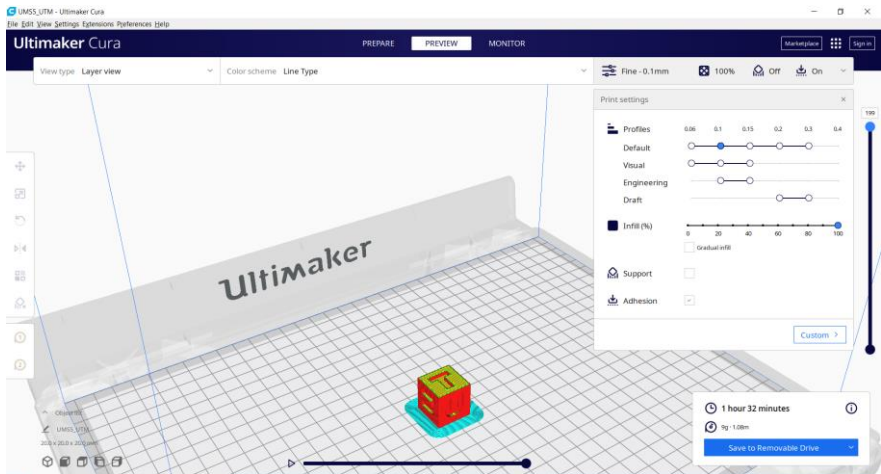


Figura 7.6. Exemplu de slice-ing al modelului 3D în Cura

Pentru schimbarea modelului de umplere în Cura se activează setarea ascunsă a modelului de umplere care va apărea în secțiunea Umplere.

Există 13 modele diferite de umplere, unele dintre ele, cele mai importante, sunt:

- **Grid:** umplutură sub formă de grilă, cu linii în ambele direcții diagonale pe fiecare strat.
- **Lines:** creează o umplutură sub formă de grilă, imprimând într-o direcție diagonală pe strat.
- **Triangles:** creează un model de umplere în formă triunghiulară.
- **Cubic:** umplere 3D de cuburi înclinate.
- **Tetrahedral:** umplere 3D de forme piramidale.
- **Concentric:** umplutura se imprimă din exterior spre centrul modelului. În acest fel, liniile de umplere nu vor fi vizibile prin pereții imprimării.
 - **Concentric 3D:** imprimeurile de umplutură din exterior spre centrul modelului, cu o înclinare peste întregul imprimeu.
 - **Zig Zag:** umplutură sub formă de grilă care se imprimă continuu într-o singură direcție diagonală.

Un exemplu de previzualizare a modelului final este arătat în figura 7.7.

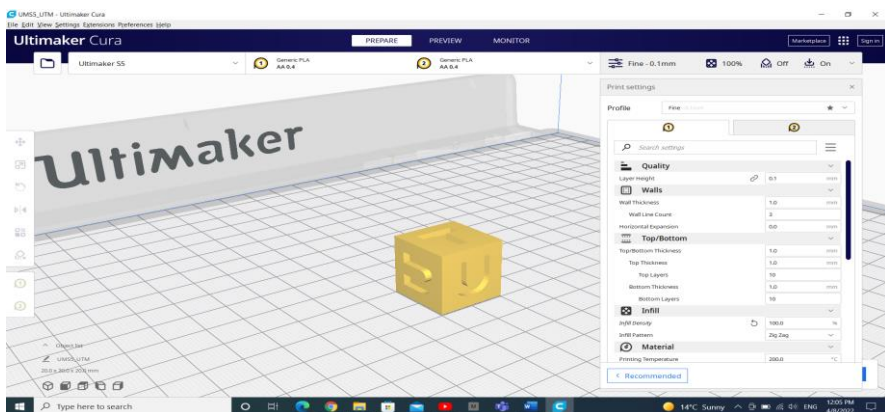


Figura 7.7. Previzualizarea modelului 3D în Cura

Modalitatea de selectare a modelului de umplere

Factorii majori care trebuie luați în considerare când este ales modelul de umplere în Cura-slicer sunt:

1. Dacă piesa va fi folosită în scopuri mecanice.
2. Dacă modelul are o suprafață mare de acoperire.

Dacă piesa nu va fi folosită ca piesă mecanică, ci mai degrabă în scopuri estetice, atunci umplutura poate să nu fie folosită. Cu toate acestea, dacă același model 3D are o suprafață mare de acoperire, atunci suprafața de acoperire va necesita un anumit sprijin pentru a fi imprimată. În astfel de cazuri, cea mai bună alegere de umplere este Concentric. Acesta utilizează material minim și este cel mai rapid de imprimat. În același timp, oferă suficient suport pentru suprafața superioară.

Dacă se dorește ca modelul să aibă o rezistență rezonabilă, chiar dacă nu va fi folosit în scopuri mecanice, atunci cea mai bună opțiune este să se selecteze un model 2D, cum ar fi Grid, Lines sau Triangles. Lines oferă cea mai mică rezistență, dar nu consumă mult material și se imprimă rapid. Grid consumă mai mult material, este

mai lent, dar oferă mai multă rezistență. Triangles oferă o rezistență mare și sarcini laterale mari. Această umplutură se utilizează când este necesară o rezistență bună a peretelui sau o structură mai lungă și mai subțire.

Dacă modelul va fi utilizat în scopuri mecanice, cea mai bună opțiune este alegerea unui model de umplere 3D, cum ar fi Cubic sau Tetrahedral. Prin aceste modele în Cura poate fi obținut un suport intern excelent și proprietăți mecanice aproape izotrope.

Generarea G-code file cu Cura

Modelul este gata de printare și rămâne doar a exporta fișierul din Cura sau pe un card SD sau să fie expediat direct la printer. Cura va converti totul din 3D STL sau OBJ într-un fișier G-code necesar imprimantei.

1. Save the 3D print file: se face click pe Save to file, Save to SD sau Send to Printre în colțul drept al ferestrei.

2. Estimate of time for 3D print: Cura va arăta timpul necesar până când va fi gata piesa printată.

3. Start the 3D print: se salvează pe cardul SD, apoi cardul SD este scos din calculator și transferat la printer. Se selectează Print și se alege fișierul după care este pornit.

Lucrarea de laborator nr. 7

Tema: EXPORTAREA MODELELOR GRAFICE

Obiectivele lucrării:

1. Familiarizarea cu conceptul de modelare 3D pentru imprimarea 3D - înțelegerea principiilor de bază ale procesului de modelare 3D și studierea caracteristicilor obiectelor compatibile cu imprimarea 3D.

2. Utilizarea aplicației Cura (sau a unei aplicații la alegere) pentru pregătirea modelelor 3D - înțelegerea interfeței aplicației Cura și configurarea corectă a setărilor. Importul și manipularea modelelor 3D (scalare, rotație, poziționare).

3. Optimizarea modelelor pentru imprimarea 3D - setarea parametrilor de imprimare (înălțimea stratului, viteza de imprimare,

densitatea umpluturii etc.). Alegerea tipului de material și analiza impactului asupra imprimării.

4. Exportul fișierelor pentru imprimarea 3D - salvarea modelelor în formatul corespunzător (G-code) pentru utilizarea pe imprimanta 3D. Înțelegerea conexiunii dintre aplicația Cura și imprimanta 3D.

5. Dezvoltarea abilităților practice - aplicarea cunoștințelor în proiectarea și pregătirea obiectelor pentru imprimarea 3D. Rezolvarea problemelor comune care apar în procesul de pregătire a modelelor pentru imprimare.

Numărul de ore necesare pentru realizare – 3 ore academice.

Scopul lucrării: obținerea cunoștințelor practice privind modelarea obiectelor pentru imprimanta 3D, utilizând aplicațiile corespunzătoare.

Sarcina lucrării: elaborați un model 3D la alegere în aplicația Cura (sau o altă aplicație specializată la algere) și pregătirea modelului 3D pentru imprimarea 3D.

Criterii de evaluare

1. Corectitudinea conversiei (10%) - verificarea corectitudinii conversiei este realizată corect, fără erori și deformare.

2. Adăugarea modelului 3D în Cura - (10%).

3. Vizualizarea modelului în Cura - verificarea modului de vizualizare.

4. Modele de umplere Cura 3D (10%) - verificarea modelului de umplere a obiectului.

5. Generarea unui G-code file cu Cura (10%) - verificarea exportului fișierului din Cura.

6. Creativitatea și complexitatea compoziției (10%) - măsurarea gradului de creativitate și complexitate a obiectului.

7. Respectarea cerințelor și optimizarea codului (10%) - verificarea dacă studentul a respectat întocmai cerințele lucrării și

evaluarea eficienței codului în utilizarea resurselor și evitarea codului redundant.

8. Estetica vizuală (10%) - analiza atractivității vizuale a compoziției, cum ar fi armonia culorilor, proporțiilor și echilibrului grafic.

9. Respectarea termenului de predare (10%) - evaluarea punctajului în funcție de punctualitate, dacă lucrarea a fost predată în termenul stabilit.

10. Evaluarea cunoștințelor (10%) - explicații privind procesul de realizare a lucrării, ceea ce poate include descrierea funcțiilor principale și a etapelor utilizate.

Întrebări de verificare a cunoștințelor

1. Care sunt formatele alternative pentru modelele de imprimare 3D?
2. Numiți tipurile de imprimante 3D.
3. Ce tehnologii de printare 3D cunoașteți?
4. Comentați tipul de format de fișier STL.
5. Ce Software de modelare 3D cunoașteți?
6. Pentru ce este utilizat Slicer 3D?
7. Care sunt cele mai utilizabile conversii?

BIBLIOGRAFIE

1. L. McCarthy, C. Reas, and B. Fry, *Getting started with p5.js: making interactive graphics in JavaScript and Processing*, First edition. in Make. San Francisco, CA: Maker Media, 2016.
2. E. Arslan, *Learn JavaScript with p5.js: coding for visual learners*. Place of publication not identified: Apress, 2018.
3. Referințele limbajului p5.js <https://p5js.org/reference/>
4. p5.js Overview <https://github.com/processing/p5.js/wiki/p5.js-overview>
5. Cornel Marin, Modelarea sistemelor mecanice <https://regielive.net/cursuri/mecanica/modelarea-sistemelor-mecanice-100377.html>
6. Physics Simulations. Erik Neumann <https://www.myphysicslab.com/>
7. Proiectare 3D <https://3dprint.capib.ro/proiectare-3d/>
8. Cura Settings Decoded by Matt Jani, 2022 <https://all3dp.com/1/cura-tutorial-software-slicer-cura-3d/>
9. Online 3D printing service <https://www.hubs.com/3d-printing/>
10. <https://openlab.bmcc.cuny.edu/makerspace/drawing-in-p5-js/>

ANEXĂ

Modelul Raportului la lucrarea de laborator

**Ministerul Educației și Cercetării
al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică
și Microelectronică**

**Raport la
lucrarea de laborator nr. 1**
Disciplina: Grafica pe calculator

Tema: Sinteza imaginii 2D

Au efectuat:

Nume Prenume
studentul gr: TI-241

A verificat:

Nume Prenume
(profesorului)

**Chișinău
2025**

Scopul lucrării: dobândirea cunoștințelor practice privind crearea și manipularea scenelor grafice 2D statice, utilizând primitivele grafice oferite de biblioteca p5.js. Aplicarea cunoștințelor teoretice în sinteza imaginilor grafice, utilizarea eficientă a funcțiilor bibliotecii p5.js și crearea imaginii grafice 2D simple.

Sarcina lucrării: elaborați un program pentru sinteza unei scene 2D statice, utilizând cel puțin 6 primitive grafice diferite cum ar fi - *arc()*, *ellipse()*, *circle()*, *line()*, *point()*, *quad()*, *rect()*, *square()*, *triangle()*; primitivele trebuie să aibă diferite atribute, lucrarea trebuie semnată (numele, prenumele, grupa) în colțul din dreapta de jos a ecranului.

Varianta #

Program realizat în p5.js:

```
//Declarăm variabilele de sistem
var Width;
var Height;
var CurrentY;
function setup() {
  Width = 400;
  Height = 734;
  createCanvas(Width, Height);
}
function draw() {
  background(220);
  CurrentY = Height - 20;
  //Realizăm baza (1 parte)
  stroke(6, 21, 131);
  for (let i = 0; i < 20; i++) {
    line(0 + 20, CurrentY, Width - 20,
CurrentY);
    CurrentY--;
  }
  // Realizăm baza (2 parte)
  stroke(15, 23, 71);
```

```

    for (let i = 0; i < 20; i++) {
        line(0 + 20 + i, CurrentY, Width - 20 - i,
CurrentY);
        CurrentY--;
    }
    CurrentY += 10;
    stroke(6, 21, 121);
    for (let i = 0; i < 550; i++) {
        line(0 + 40, CurrentY, Width - 40,
CurrentY);
        CurrentY--;
    }
    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);
    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);

```

```

    }
}

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);
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);
    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);
    }
    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



Concluzii

În urma realizării lucrării de laborator am dobândit cunoștințe practice esențiale privind sinteza scenelor grafice 2D statice, utilizând primitivile grafice simple puse la dispoziție de biblioteca p5.js. Prin utilizarea eficientă a primitivelor grafice (puncte, linii, dreptunghiuri, cercuri etc.) și a funcțiilor de stilizare a fost creată

scena grafică, am observat importanța fiecărei primitive în construirea formelor și obiectelor de bază într-o scenă grafică.

Am aprofundat cunoștințele utilizând tehnicile de modificare a atributelor grafice, cum ar fi culoarea, conturul, transparența și umplerea, elemente esențiale în definirea esteticii unei compoziții. De asemenea, am învățat să afișez și să stilizez textul în mod grafic, integrându-l armonios în cadrul scenei, ceea ce reprezintă un pas important în construirea imaginilor.

Experiența obținută în urma realizării lucrării de laborator a contribuit la consolidarea cunoștințelor teoretice și a abilităților practice necesare pentru lucrul cu grafica pe calculator.

CUPRINS

I.	Noțiuni generale privind bibliotecile grafice p5.js	9
	1.1. Crearea primitivelor grafice 2D simple	12
	1.2. Atribute grafice	20
	1.3. Variabile de sistem în p5.js	24
	1.4. Formatele grafice	255
	Lucrarea de laborator nr. 1	27
II.	Grafica vectorială.....	35
	2.1. Caracteristicile grafice vectoriale	35
	2.2. Formatul SVG.....	36
	2.3. Primitive grafice 2D în <svg>	37
	2.4. Integrarea grafice vectoriale în p5.js Web Editor	45
	2.5. Software gratuit pentru grafica vectorială	46
	Lucrarea de laborator nr.2	48
III.	Transformări geometrice 2D	54
	3.1. Translația	54
	3.2. Rotația	57
	3.3. Scalarea	60
	Lucrarea de laborator nr.3	64
IV.	Utilizarea obiectelor 3D în scenele statice	72
	4.1. Fișiere *.OBJ	72
	4.2. Descrierea obiectelor și metodelor principale în lucrul cu fișierele *.OBJ în editorul online p5.js	73
	4.3. Crearea scenei statice 3D	75
	Lucrarea de laborator nr. 4	79
V.	Realizarea scenei dinamice în 3D.....	86
	5.1. Introducere în realitatea augmentată	86
	5.2. Platforma Artivive	88
	Lucrarea de laborator nr. 5	91
VI.	Modelarea proceselor 3D dinamice	94
	6.1. Crearea primitivelor grafice 3D simple	986
	Lucrarea de laborator nr. 6	100
VII.	Crearea modelelor pentru imprimarea 3D.....	105

7.1. Tehnologii de printare 3D	105
7.2. Formatul fișierelor STL.....	112
7.3. Slicer 3D.....	114
7.4. Formatul STL.....	115
7.5. Deschiderea fișierului STL.....	116
7.6. Crearea fișierului STL.....	116
7.7. Alternativa fișierului STL	117
7.8. Limbajul GCode	118
Lucrarea de laborator nr. 7	126
Bibliografie.....	129
Anexă. Modelul Raportului la lucrarea de laborator	130

Grafica pe calculator

Îndrumar metodic pentru lucrările de laborator

Autori: Lilia Rotaru
Mariana Oșovschi
Viorel Cărbune
Adriana Ursu

Redactor E. Balan

Bun de tipar	Formatul hârtiei 60x84 1/16
Hârtie ofset. Tipar RISO	Tirajul 50 ex.
Coli de tipar 8,25	Comanda

MD-2004, Chișinău, bd. Ștefan cel Mare și Sfânt, 168, UTM
MD-2045, Chișinău, str. Studenților, 9/9, Editura „Tehnica-UTM”