UNIVERSITATEA TEHNICĂ A MOLDOVEI

FACULTATEA CALCULATOARE, INFORMATICĂ, MICROELECTRONICĂ

Rotaru Lilia, Oșovschi Mariana,
Cărbune Viorel, Ursu Adriana

GRAFICA PE CALCULATOR

ÎNDRUMAR METODIC PENTRU LUCRĂRI DE LABORATOR

Chișinău
Editura „Tehnica-UTM” 2025

Modalitatea de evaluare a cunoştinţelor şi rezultatelor activităţii practice

Pentru fiecare lucrare de laborator se va elabora un raport în baza modelului prezentat în Anaxa 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 scrise 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 criterii de evaluare prezentate în fiecare lucrare de laborator.

Studentul trebuie să respecte termenul limită de două săptămîni pentru prezentarea lucrării de laborator. Pentru fiecare săptămână întârziere profesorul este în drep să depuncteze studentul pentru lucrarea respectivă.

Criterii de evaluare	Descriptori				Ponderea criteriului de evaluare în nota finală acordată candidatului
	Nivel maxim (nota 9.00-10)	**Nivel mediu** (nota 7.00-8.99)	**Nivel minim** (nota 5.00-6.99)	**Nivel insuficient** (nota <5.00)	
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 facă expunerea raportului	0.3

Criterii de evaluare	Descriptori				Ponderea criteriului de evaluare în nota finală acordată candidatului
	Nivel maxim (nota 9.00-10)	**Nivel mediu** (nota 7.00-8.99)	**Nivel minim** (nota 5.00-6.99)	**Nivel insuficient** (nota <5.00)	
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 a bibliotecii grafice p5.js

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

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

Creare 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 prezentate în figura 1.

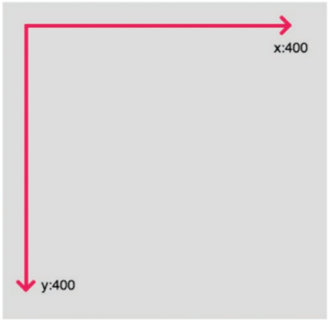


Figura 1. Sistemul de coordinate în mediul p5.js

Pentru crearea unui program simplu în p5.js trebue să utilizăm 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 a mediului de lucru, cum ar fi dimensiunea și culoarea ecranului, încărcarea fișierelor multimedia la lansarea programul, cum ar fi imagini și fonturi. 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() – crează un element canvas (pînză) și setază dimensiunea acestuia în pixeli. Această metodă ar trebui apelată o singură dată la începutul programului. Apelarea **createCanvas** de mai multe ori într-un cod va avea ca rezultat un comportament foarte imprevizibil. Dacă doriți mai mult de o pânză de desen, puteți utiliza createGraphics.

Sintaxa:

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

unde:

w – număr lățimea pînzei;

h – număr înnățimea pînzei;

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

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

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

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

Sintaxa funcției este:

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

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

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

Sintaxa:

```
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 culurii 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) (obțional);

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 fon;

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 primiteve grafice sunt primitivele grafice 2D, în figura 1.1 arătată corespunderea punctelor figurilor geometrice și parametrilor care trebuie indicate ca argumentele funcțiilor în codul programului.

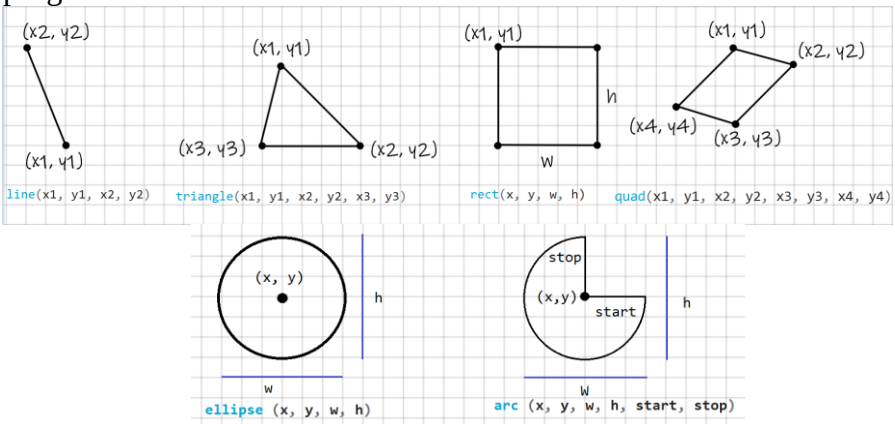


Figura 1.1. Relația dinte punctele geometrice și parametrii funcțiilor P5.js

Funcția arc(): desenează un arc pe ecran. Dacă este apelat doar cu ***x, y, w, h, start*** și ***stop***, arcul va fi desenat și umplut ca un segment de cerc deschis. Dacă este specificat parametru *mod*, arcul va fi umplut ca un semicerc deschis (OPEN), un semicerc închis (CHORD) sau un segment circular închis (PIE). Sursa poate fi schimbată funcția **ellipseMode()**. Diferența dintre aceste regimuri de desenare este arătată în figura 1.3.

Arcul întotdeauna este desenat în sensul acelor ceasornicului punctul de început (start) și sfârșit (stop) pot folosi constantele p5.js conform valorilor indicate în circumferința prezentată în figura 1.2. Dacă punctele start și stop cad în același loc, se va desena o elipsă (cerc) completă. Axa *Y* crește în direcția 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) – înnălțimea arcului;

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

stop (număr întreg) – unghiul de svîrșit a arcului;

mode (constantă) – parametru care determină modul în care este desenat arcul. COORD, PIE sau OPEN (opțional);

detail (număr întreg) – parametru opțional numai pentru modul WebGL;

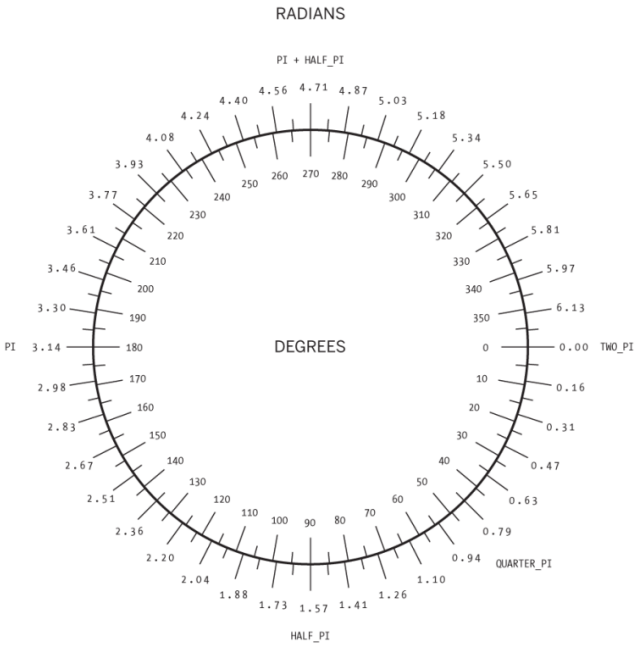


Figura 1.2. Relația dintre constante, radiani și grade în p5.js [10]

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 prezentat în figură 1.3

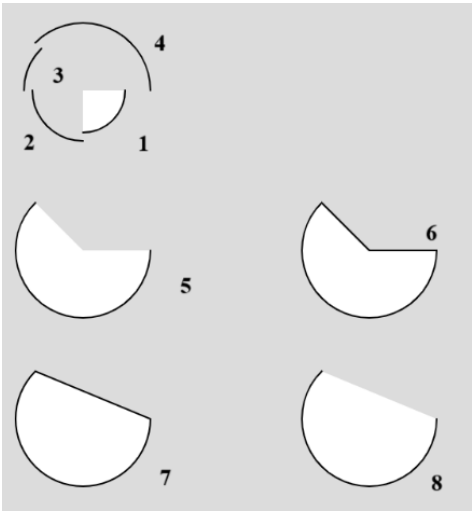


Figura 1.3. Exemple de desenare a arcurilor

Funcția ellipse(): desenează o elipsă (oval) pe ecran. Elipsa cu lățime și înălțime egale este un cerc. În mod implicit, primii doi parametri specifică coordonatele centrului figurei, 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 figurei;

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

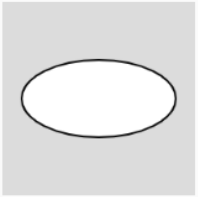
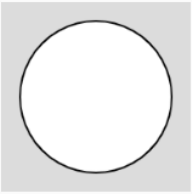
w (număr întreg) – lățimea elipssei;

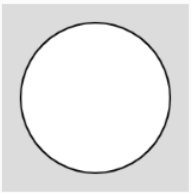
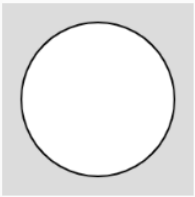
h (număr întreg) – înnălțimea elipsei;

detail (număr întreg) – parametru 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

Programul	**Rezultatul**
function setup() { createCanvas(100, 100); } function draw() { background(220); **ellipse(50, 50, 80, 40); }**	
function setup() { createCanvas(100, 100); } function draw() { background(220); **ellipse(50, 50, 80);** }	

Programul	Rezultatul
function setup() { createCanvas(100, 100); } function draw() { background(220); **ellipse(50, 50, 80, 80);**}	
function setup() { createCanvas(100, 100); } function draw() { background(220); **circle (50, 50, 80);** }	

Funcția circle(): desenează un cerc pe ecran. Această funcție este un caz particular al funcției **elipse()**, unde lățimea și înălțimea elipsei sunt aceleași. Înălțimea și lățimea elipsei corespund diametrului cercului. În mod implicit, primii doi parametri stabilesc coordonatele centrului cercului, al treilea - diametrul cercului

Sintaxa:

```
circle(x, y, d);
```

unde:

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

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

d (număr întreg) – diametrul cercului;

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

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);
coordonate_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 formei 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 primului punct;
z2 – coordonata z punctului doi;
z3 – coordonata z punctului trei;
z4 – coordonata z punctului patru;

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

Al cincilea, al șaselea, al șaptelea și al optulea parametri, dacă sunt specificati, 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:

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

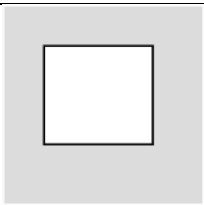
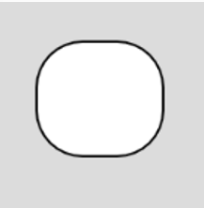
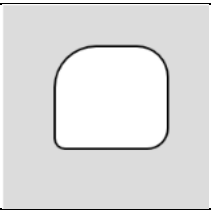
unde:

x – coordonata x a dreptunghiului;
y – coordonata y a dreptunghiului;
w – lățimea dreptunghiului;
h – înnălțimea dreptunghiului;
tl – raza de rotungire a colțului stânga-sus (opțional);
tr – raza de rotungire a colțului dreapta-sus (opțional);
br – raza de rotungire a colțului dreapta-jos (opțional);
bl – raza de rotungire 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 raza de rotungire a tuturor colțurilor vă fi egală (*tl= tr= br= bl*), atunci poate fi indicat doar un parametru.

Exemple de utilizare sunt aduse în tabelul 1.2.

Tabelul 1.2 Exemple de utilizare a funcției rect()

Programul	**Rezultatul**
function setup() { createCanvas(100, 100); } function draw() { background(220); **rect(20, 20, 55, 50);** }	
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);**	
function setup() { createCanvas(100, 100); } function draw() { background(220); **rect(25, 20, 55, 50, 20, 15, 10, 5);** }	

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

Parametrii – patru, cinci, șase și șapte, dacă sunt specificati, definesc raza pentru colțurile din stânga sus, din dreapta sus, din dreapta jos și, respectiv, din stânga jos. Parametrul razei colțului este setat la valoarea razei specificate anterior în lista de parametri.

Sintaxa:

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

unde:

x – coordonata x a pătratului;

y – coordonata y a pătratului;

s – mărimea laturei pătratului;

tl – 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. Argumenții funcției specifică coordonatele primului vîrf, coordonatele vîrfului doi, iar ultimele două argumente indică coordonatele vîrfului al treilea al treiunghiului.

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 at fi culoarea figurii, grosimea și culoarea liniei, tipul hașurării, modul de conexiune a liniilor reprezintă atributele grafice simple. În biblioteca p5.js sunt 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 umplute (colorate) cu culoarea indicate ca parametru funcției. Această culoare este specificată în termeni de culoare RGB sau HSB, în funcție de **colorMode()** curent. (Spațiul de culoare implicit este RGB, fiecare valoare variază de la 0 la 255).

Sintaxa:

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

unde:

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

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

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

alpha (număr întreg) – (opțional);

value (un string) – string care indică culoarea;

gray (număr întreg) – valoarea culorii sure;

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

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 nuanței în raport cu gama de culori curentă;

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

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

alpha (număr întreg) – (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:rușii, verde, albastră și alpha;

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

În afara culorii funcția stroke poate specifica următoatele:

```
strokeCap();
strokeJoin();
strokeWeight();
```

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

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

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

Diferența dintre aceste moduri este prezentată în figura 1.4.

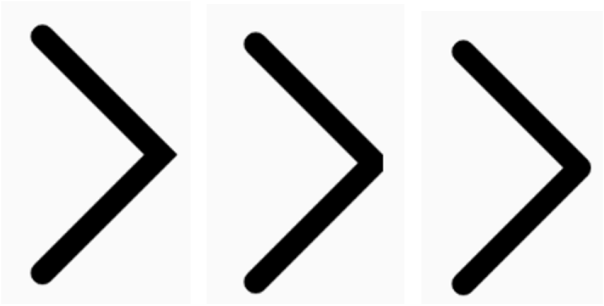


Figura 1.4. Parametrii funcției strokeJoin

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

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

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

Sintaxa:

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

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 *x2* și *y2* nu

sunt specificate, alinierea liniei de bază este implicită, ceea ce înseamnă că textul va fi desenat în sus de la *x* și *y*.

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 un cuvânt cheie care indică compilatorului că urmează definirea variabilei.

Variabilele pot fi locale sau globale. Ciclul de viață a 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, care determină modul în care sunt stocate și afișate pe ecran sau la imprimare datele.

Formatul fișierului este de obicei indicat în numele său, ca o parte separată printr-un punct (de obicei, această parte se numește extensia numelui fișierului).

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

Formatele grafice se clasifică după:

- ✓ tipul datelor stocate (raster, vector și forme mixte);
- ✓ în funcție de cantitatea admisă de date;
- ✓ parametrii imaginii;
- ✓ modul de stocare a paletei de culori;
- ✓ metoda de compresie a datelor;
- ✓ prin metode de organizare a fișierelor (text, binar);

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

Lucrarea de laborator 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 în 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, care sunt utilizate în construirea scenelor vizuale în grafica 2D.

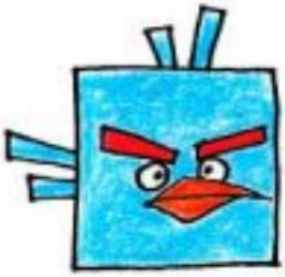
Numărul de ore necesare pentru realizare – 4 ore academice.

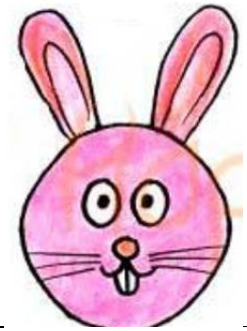
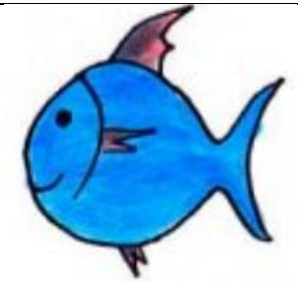
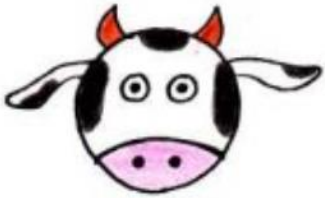
Scopul lucrării: dobândirea cunoștințe practice în crearea și manipularea scenelor grafice 2D statice, utilizând primitivele grafice oferite de biblioteca p5.js. Aplicarea cunoștințelor teoretice în sinteza imaginilor grafice, utilizerea eficientă a funcțiilor bibliotecii p5.js și crearea imaginii grafice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

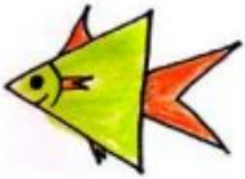
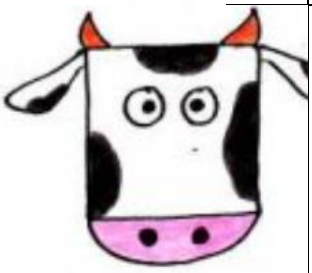
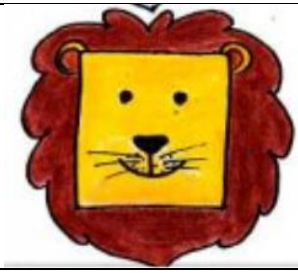
Sarcina lucrării:

1. Elaborați un program pentru sinteza unei scene 2D statice utilizîng cel puțin 6 primitive grafice de diferite tipuri cum ar fi - *arc(), ellipse(), circle(), line(), point(), quad(), rect(), square(), triangle()*, primitivele trebuie să fie cu diferite atribute, lucrarea trebuie semnată (numele prenumele grupa) în colțul dreapta jos a ecranului. În program creați personajul conform variantei indicate de profesor luînd în considerație cerințele indicate mai sus. Variantele sunt prezentate în tabelul 1.3. Crearea acestor imagini pe pași poate fi consulta pagina: https://ja-rastu.ru/raskraski/uroki/1079-poshagovoe-risovanie-zhivotnyh-iz-geometricheskih-figur.html

Tabelul 1.3 Variantele pentru realizarea lucrarii de laborator

Var.	Personajul	Var.	Personajul
1		11	
2		12	

Var.	Personajul	Var.	Personajul
3		13	
4		14	
5		15	
6		16	

Var.	Personajul	Var.	Personajul
7		17	
8		18	
9		19	
10		20	

Exemplu:

În exemplu ce urmează este prezentat un cod în care este realizat un personaj în baza condițiilor propuse în sarcină 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 gură
      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 mâinele
      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 picioare
      fill(0);
      rect(180, 290, 10, 50);
      rect(210, 290, 10, 50);
// Scriem numele prenumele și grupa în colțul din dreapta jos
      fill(0);
      textSize(12);
      text("Nume Prenume, grupa", width - 160, height -
10); }
```

Rezultatul execuției programului este arătat în figura 1.5.

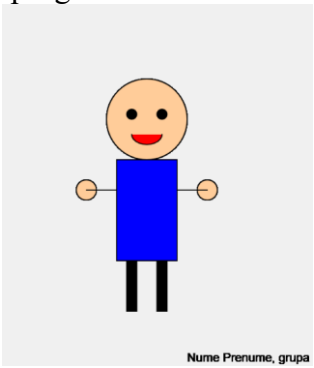


Figura 1.5. Rezultatul execuției codului

Criterii de evaluare:

1. **Corectitudinea codului (10%)** – verificarea dacă codul este scris corect, fără erori de sintaxă și funcționare.
2. **Utilizarea primitivelor grafice (10%)** – evaluarea utilizării corecte și variatea 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 dacă studentul a respectat întocmai cerințele 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** (dacă este aplicabil) - dacă lucrarea include cerințe interactive, cum ar fi răspunsuri 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%)** – explicațiile oferite despre 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 primitive grafice simple.
2. Cum poate fi realizată modificarea atributelor de afișare ale primitivelor grafice?
3. Cum poate fi scris textul în mod grafic?
4. Numiți formate standard pentru imagini.
5. Enumerați primitive 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 a afișarea textul 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 pentru imagini raster.
14. Care sunt diferențele între formatele JPEG, PNG și GIF?
15. Ce format de imagine este cel mai potrivit pentru transparență și de ce?

Bibliografia

[1] L. McCarthy, C. Reas, and B. Fry, *Getting started with p5.js: making interaktive 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/

Anexa 1

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 – 2024

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

Sarcina lucrării:

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

Varianta #

Program realizat în p5.js:

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

```
      for (let i = 0; i < 20; i++) {
        line(0 + 20 + i, CurrentY, Width - 20 - i,
CurrentY);
        CurrentY--;
      }
      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 prgramului:

Concluzie:

În urma realizării acestei lucrări de laborator, am dobândit cunoștințe practice esențiale pentru sinteza scenelor grafice 2D statice, utilizând primitivele 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 stilizăz textul în mod grafic, integrându-l armonios în cadrul scenei, ceea ce reprezintă un pas important în construirea imaginilor.

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