UNIVERSITATEA TEHNICĂ A MOLDOVEI

Rotaru Lilia, Oșovschi Mariana,
Cărbune Viorel, Ursu Adriana

GRAFICA PE CALCULATOR

ÎNDRUMAR METODIC PENTRU LUCRĂRILE DE LABORATOR

Chișinău
2025

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;

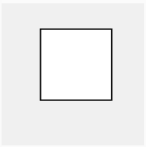
z – un număr întreg care indică deplasarea pe axa z pozitivă;

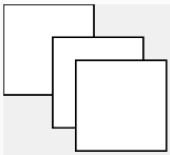
vector – p5.Vector în baza căruia se realizează deplasarea.

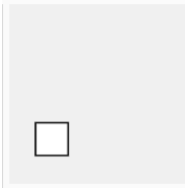
Exemple de utilizare a funcției translate() sunt inserate în tabelul 3.1.

Tabelul 3.1. Exemple de utilizare a funcției translate()

Programul	***Rezultatul***

`translate(30, 20);`
`rect(0, 0, 55, 55);`
`//echivalent cu`
`//rect(30,20,55,55);`
	

`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
	

`function draw() {`
`background(200);`
`rectMode(CENTER);`
`translate(width/2, height/2);`
`translate(p5.Vector.fromAngsle (millis()/1000, 40));`
`rect(0, 0, 20, 20); }`
	

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);
```

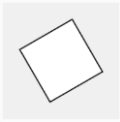
unde:

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

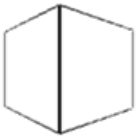
Exemple de utilizare a funcțiilor sunt date în tabelul 3.2.

Tabelul 3.2. Modul de utilizare a funcției rotate

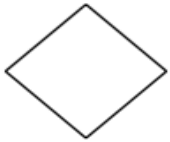
Programul	***Rezultatul***

translate(width / 2, height / 2);
rotate(PI / 3.0);
rect(-26, -26, 52, 52);
	

function draw() {
background(255);
rotateX(millis() / 1000);
box();
	

function draw() {
background(255);
rotateY(millis() / 1000);
box();
	

Programul	*Rezultatul*

function draw() {
background(255);
rotateZ(millis() / 1000);
box();
	

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 ere 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);

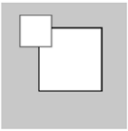
z – procentul pentru scalarea obiectului de-a lungul axei ***z*** (numai webgl) (opțional);

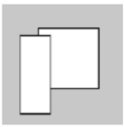
scales – procentul pe axe pentru scalarea obiectului.

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.

Tebelul 3.3. Modul de utilizarea a funcției scale()

Programul	***Rezultatul***

rect(30, 20, 50, 50);
scale(0.5); //uniform
rect(30, 20, 50, 50);
	

rect(30, 20, 50, 50);
scale(0.5, 1.3); //neuniform
rect(30, 20, 50, 50);
	

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 (fraimuri) 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.

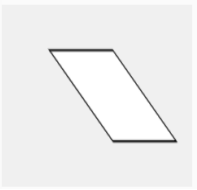
Sintaxa:

```
shearY(angle);
```

unde:

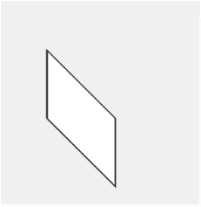
angle – unghiul de deplasare specificat în radiani sau grade, în funcție de unghiul curent din angleMod.

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



```
function draw() {
  background(240);
   translate(width/4, height / 4 );
 shearX(PI/4.0);
  rect(0 , 0 ,50, 50);
}
```

a) funcția shearX()



```
function draw() {
  background(240);
   translate(width/4, height / 4 );
 shearY(PI/4.0);
  rect(0 , 0 ,50, 50);
}
```

b) funcția shearX()

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:** foloserea rotației pentru animarea mișcării pendulului.
- **Sugestie:** să se utilizeze **sin()** pentru o mișcare lentă și naturală.

Varianta 2. Sistemul solar: să se construiescă 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 construiească 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 mișcarii (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 secundele.

- **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 translatate 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.

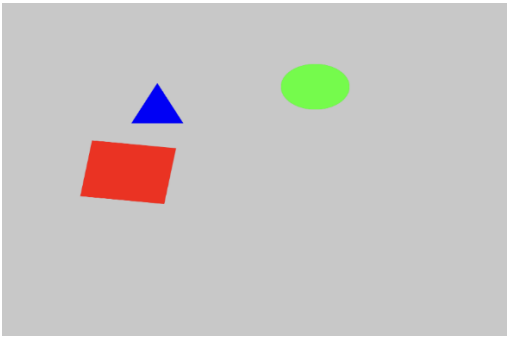


Figura 3.4. Scenă dinamică 2D

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?

BIBLIOGRAFIE

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/

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 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 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.