Multimedia

lect. dr. Cristian Ioniță

Structura curs

Obiectiv

Însușirea elementelor teoretice și practice care să permită dezvoltarea de aplicații multimedia

Cuprins

1. Noțiuni generale
2. Imagine și animație
3. Sunet
4. Video

Evaluare: proiect și lucrare scrisă

Contact

- Cristian Ioniță – cristian.ionita@softmentor.ro
- Materiale: http://ase.softmentor.ro

I. Noțiuni generale

1. Conceptul de multimedia

2. Clase de aplicații multimedia

3. Condiții hardware / software

4. Multimedia în context WEB

1. Conceptul de multimedia

Multimedia din punct de vedere informatic este:

- o combinație de text, imagine, sunet, animație, video
- accesibilă utilizatorului prin intermediul sistemului de calcul

Elemente care au stat la baza dezvoltării conceptului de multimedia:

- conversia analog digital
- compresia datelor

Conversia analog - digital

Resursele utilizate trebuie convertite în format digital pentru:

- stocare
- procesare
- includere în aplicații

Resursele sunt convertite în format analog la redare

Compresia

- apariția algoritmilor specifici (cu pierdere de informație)
- exemple: JPEG, MPEG Audio / Video
- pierdere de informație controlată
  - exploatează limitările percepției umane
- proces asimetric

Tehnologii utilizate

- periferice specializate pentru captură imagine / sunet / video
- medii de stocare de mare capacitate
- tehnologii de comunicare la distanță cu bandă largă
- procesoare specializate pentru compresie / decompresie la nivel hardware

Aplicație multimedia

- componentele sunt accesibile prin intermediul unui sistem de calcul
- datele utilizate sunt în format digital (NU analogic)
- elementele sistemului sunt integrate într-o interfață unitară
- grad ridicat de interacțiune cu utilizatorul

Modalități de dezvoltare

Multimedia authoring

- includ componente preprogramate ce permit recunoașterea mai multor formate de resurse multimedia, instrumente pentru generarea animațiilor, pentru implementarea conceptelor de hypertext, hypermedia
- accentul cade pe scenariul de derulare a aplicației și pe sincronizarea resurselor
- bazat pe concepte precum axa timpului / carte / diagrame de flux
- Exemple: Flash, PowerPoint

Multimedia programming:

- accentul se pune pe procesarea directă a resurselor prin intermediul unui limbaj de programare și a unor biblioteci specializate

2. Clase de aplicații multimedia

Criterii multiple de clasificare

Cele mai importante criterii

- domeniu de utilizare
- grad de interacțiune
- localizarea componentelor

Clasificare în funcție de domeniu

- **Economie** – vizualizarea datelor
- **Educație** – aplicații de e-learning, enciclopedii
- **Publicitate** – aplicații de prezentare
- **Medicină** – procesarea și vizualizarea interactivă a datelor medicale
- **Industrial** –instrumente de proiectare asistată
- **Entertainment** – jocuri, realitate virtuala
- **Sisteme informatice geografice GIS** – sisteme pentru vizualizarea datelor geo-spațiale
- **Comunicații** – aplicații de tip videoconferință

Alte criterii

Grad de interacțiune

- aplicații interactive
- prezentări statice

Localizarea componentelor

- locale
- la distanță

3. Precondiții hardware / software

Echipamente hardware pentru achiziție / procesare:

- imagine
- sunet
- video

Componente software necesare pentru redarea conținutului multimedia

Hardware specializat

Dispozitive periferice pentru achiziția de **imagini fixe**:

Scanner

- transformă informația luminoasă în informație electrică, iar ulterior aceasta este convertita și salvată sub formă digitală
- tipuri: flatbed / rotativ

Aparat foto digital

- folosește lentile asemănătoare aparatului foto clasic și un senzor digital pentru transformarea informației luminoase în informație electrică

Hardware specializat

Dispozitive periferice pentru achiziția de **sunet**:

- placa de sunet: convertor analog – digital și digital – analog

Dispozitive periferice pentru achiziția de **imagini video**:

- placa de achiziție și numerizare video: preluare și numerizare fluxuri audio - video
- camera video digitală: captură directă în format digital (similar aparat foto digital)
- placa TV tunner: preia semnal TV, decodifică și numerizează semnalul

Software specializat

Drivere

- Asigură controlul și comunicarea cu perifericele specializate
- Specifice pentru dispozitivul și sistemul de operare
- Permit sistemului de operare să ofere aplicațiilor o interfață uniformă

Extensii ale sistemului de operare

- Biblioteci specializate pentru controlul resurselor multimedia
- Instrumente de bază pentru redarea conținutului multimedia

Software specializat

Produse software specializate pe medii de comunicare

Achiziție și prelucrare de **imagini**

- Permit prelucrarea imaginilor în format raster sau vectorial
- Exemple: Adobe PhotoShop, GIMP, Corel Draw, Adobe Illustrator

Achiziție și prelucrare de **sunet**

- Exemple: Adobe Audition, Audacity, Sony Sound Forge

Achiziție și editare **video**

- Exemple: Adobe Premiere, Blender, Sony Vegas, DaVinci Resolve

Software specializat

Produse software pentru **creație și redare de aplicații multimedia**

Exemple

- Browser web
- Flash
- PowerPoint

4. Multimedia în context WEB

Se dezvoltă odată cu apariția standardului HTML5

Elemente multimedia suportate

- Text
- Imagini
- Animație
- Grafică raster – elementul *canvas*
- Grafică vectorială – SVG
- Sunet – elementul *audio + Web Audio API*
- Video – elementul *video*
- Grafică 3D – WebGL

Avantaje

Arhitectură client-server

Client – **Web Browser**

- Trimite cererile către server
- Gestionează afișarea conținutului multimedia și interacțiunea cu utilizatorul

Server – **Web Server**

- Preia cererile de la Web Browser
- Trimite conținutul (static sau generat dinamic) către browser

Comunicare – protocolul **Hypertext Transfer Protocol**

- Protocol text
- Mesaje de tip *request* și *response*
- Mesajele sunt trimise prin intermediul protocolului TCP/IP

HTTP Request

Format: comanda (GET / POST / …)/ parametri / eventual conținut

Cereri inițiate de către browser:

- Introducere adresă de către utilizator
- Navigare către pagină nouă
- Obținere resurse
- Solicitat din cod

Exemplu:

GET /index.html HTTP/1.1

Host: www.test.com

User-Agent:Mozilla/5.0

HTTP Response

Format: status (200 / 404 / 500 /…)/ parametri / conținut

Răspunsul este generat:

- Static (conținutul este preluat din fișier)
- Dinamic (conținutul este generat de către un program)

Exemplu:

HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 131

<html>
…
</html>

Limbaje utilizate

HTML - HyperText Markup Language

- Structura documentului
- Conținutul (de tip text)
- Referințe la alte resurse

CSS - Cascading Style Sheets

- Descrie modalitatea de prezentare a conținutului

JavaScript

- Manipularea dinamică a conținutului și a modalității de prezentare
- Gestionează interacțiunea cu utilizatorul

DOM – Document Object Model

Aplicația este reprezentată intern sub formă de arbore

Tipuri de noduri

- Elemente
- Text

Nodurile - create din HTML sau JavaScript

Conțin:

- atribute
- parametri de stil
- funcții JavaScript

Limbajul HTML

Tag

- Instrucțiunea de creare a unui nod
- Format: ***<test>****conținut****</test>***
- Conținutul poate fi:
  - Text
  - Alte tag-uri

Atribut

- Perechi denumire – valoare atașate nodurilor
- <test **nume1="valoare1" nume2="valoare2"**>conținut</test>

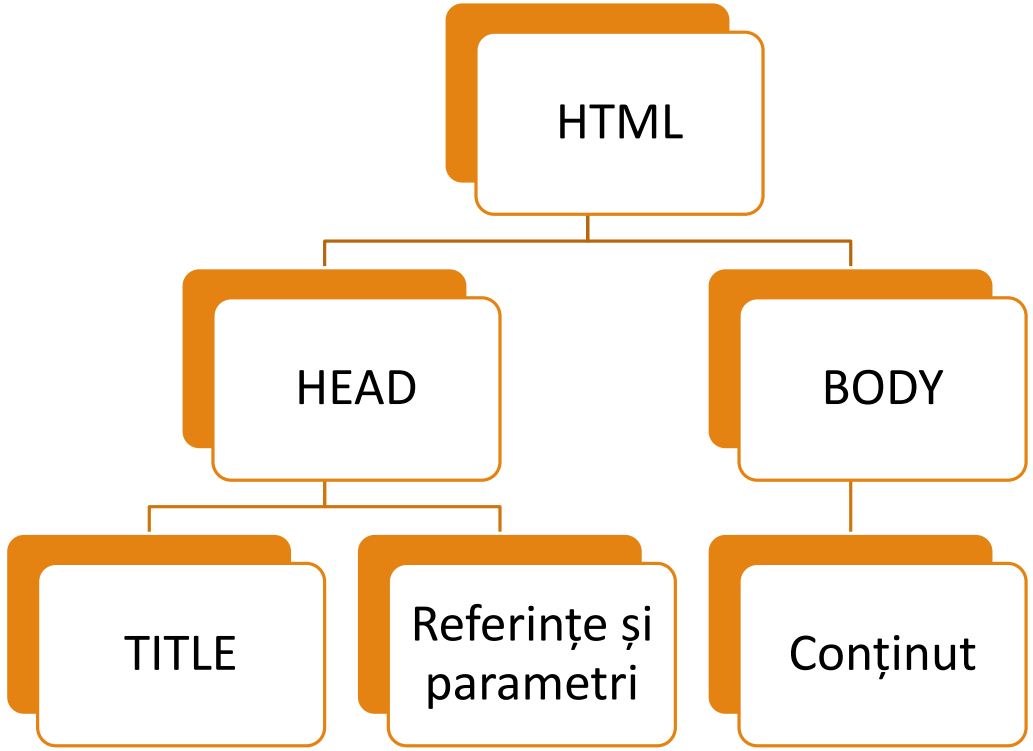
Structura unui document HTML

<!DOCTYPE html>

<html>

- <head>
  - <title>titlu</title>
  - *referințe și parametri*
- </head>
- <body>
  - *conținut*
- </body>

</html>



Elemente de tip *block* și *inline*

	Pe linie nouă	Lățime	Înălțime	Conținut
Block	Mereu	• 100% din lățimea părintelui • Poate fi setată	• Înălțimea conținutului • Poate fi setată	Elemente de tip *block* și / sau *inline*
Inline	Doar dacă nu mai are loc pe linia curentă	• Lățimea conținutului • NU poate fi modificată	• Înălțimea conținutului • NU poate fi modificată	Doar elemente *inline*

Tipuri de elemente

Text

- Formatare: ***h1*** – ***h6***, ***p***, ***pre***, ***div,*** *em, strong, span, br, a*
- Liste: ***ul***, ***ol***, *li*
- Tabele: ***table***, *thead, tbody, tfoot, tr, th, td*

Formulare: ***form***, *input, select, option*

Imagine raster

- *img, canvas*

Imagine vectorială: *svg*

Audio și video: *audio,* ***video***

Cascading Style Sheets

Modificarea modului de afișare a elementelor din DOM

Format dintr-o serie de reguli de forma:

```
selector
{
    proprietate 1 : valoare 1;
    proprietate 2 : valoare 2;
    ....
}
```

Utilizare CSS în HTML

Tag ***style*** – *stiluri interne*

```
<style type="text/css">
    th { font-weight: bold; }
</style>
```

Tag ***link*** – *stiluri externe*

```
<link rel="stylesheet" type="text/css" href="test.css" />
```

Atribut ***style*** – *stiluri inline*

```
<p style="color: red; font-size: 12pt;">...</p>
```

Selectori

Identifică nodurile din arbore asupra cărora se vor opera modificările

Selectori simpli:

- Element: **TIP_ELEMENT**
- Identificator: **#ID**
- Clasă: **.NUME_CLASĂ**

Alți selectori:

- Universal: *****
- Atribut: **selector[atribut="valoare"]**
- Nod copil sau descendent: **selector > selector** sau **selector selector**

Grupare: **selector, selector**

Combinare selectori: **TIP_ELEMENT#ID**, **TIP_ELEMENT.NUME_CLASĂ**

Reguli de aplicare

Moștenire

- Propagarea valorilor de la un nod părinte la nodurile copil
- Nu toate proprietățile sunt moștenibile

Cascadare

- Mecanismul de alegere aplicat în cazul în care avem mai multe valori în conflict
- Reguli de cascadare
  - Importanță: reguli marcate ca *!important*
  - Specificitatea selectorului
  - Ordinea în fișierul sursă

Formatare text și culori

Proprietăți:

- *font-family* – denumire font sau familie (ex: sans-serif, monospace, Arial, Tahoma, …)
- *font-size* – dimensiune caractere (ex: small, medium, 12pt, 0.8em, …)
- *font-weight* – grosime caractere (ex: normal, bold, 400, 700, …)
- *font-style* – tip caractere (ex: normal, italic, oblique, …)
- *color*, *background-color* – culoare text sau fundal (ex: red, #EE0000, #E00, rgb(238, 0, 0), …)

Unități de măsură:

- relative: %, em, ...
- absolute: pt (1 / 72 inch), px (1 / 90 inch), cm, ….

Exemplu

body { font-family: tahoma; font-size: 11pt; color: blue; *}*

Setare margini și dimensiuni

Setare dimensiuni:

- *width*
- *height*
- *min/max-width/height*
- *overflow:*
  - ***visible*** – întreg conținutul este afișat, posibil în afara celulei
  - *hidden - conținutul din afara celulei nu este afișat*
  - *scroll – hidden + afișare bară de scroll*
  - *auto – afișează bara de scroll doar dacă este necesară*

Proprietățile *width* și *height* se referă doar la dimensiunea secțiunii *content* (fără *padding / border / margin*)

Setare margini și dimensiuni

Setare margini:

- padding sau margin / -top / -bottom / -left / -right
  - *padding*:
    - *3px 4px 5px 6px* (top, right, bottom, left) sau
    - *2px 3px* (top = bottom = 2px, right = left = 3px) sau
    - *3px* (toate laturile)
  - *padding-top: 5px* (setare pentru o singură latură)
  - similar pentru *margin*
- border / -top / -bottom / -left / -right / -width / -style / -color
  - *border: 2px dotted green*
  - *border-style: solid*
  - *border-bottom-width: 3px;*

Controlul poziționării

Control layout:

- display: none / inline / block
- float: none / left / right
- clear: none / left / right / both

Poziționare în cadrul paginii

- top, right, bottom, left, z-index
- position:
  - static: poziționare normală, proprietățile (top, …) sunt ignorate
  - relative: poziționare normală, proprietățile (top, …) sunt respectate
  - absolute: nu participă la poziționarea normală; poziționat față de primul părinte cu poziționare absolută
  - fixed: nu participă la poziționarea normală; poziționat față de fereastra browser-ului

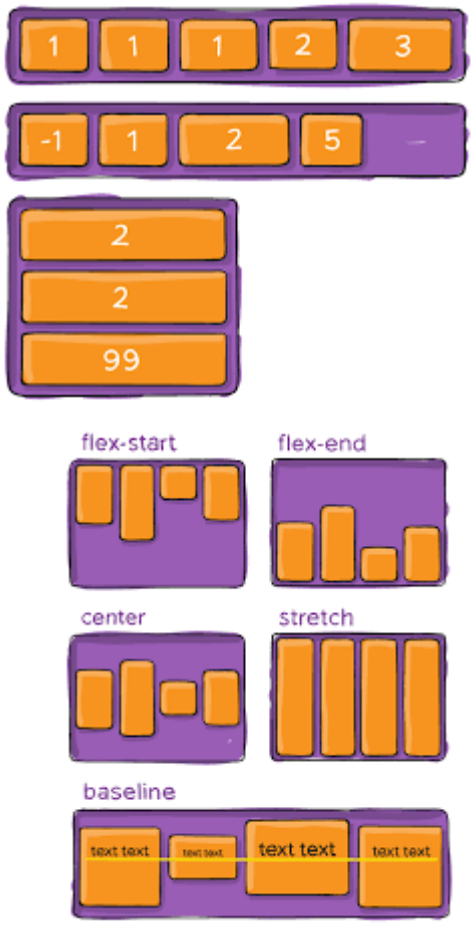
Flexbox (Opțional)

Sistem de layout utilizat pentru alinierea conținutului bazat pe un **model 1D** (o singură axă principală)

- Permite poziționarea atât pe verticală (linii) cât și pe orizontală (coloane)
- Proprietăți la nivelul container-ului părinte:
  - **display: flex;**
  - flex-direction, flex-wrap, justify-content, align-items, ...
- Proprietăți la nivelul elementelor copil:
  - order, flex-grow/shrink/basis, align-self

Resurse:

- https://css-tricks.com/snippets/css/a-guide-to-flexbox/
- https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox



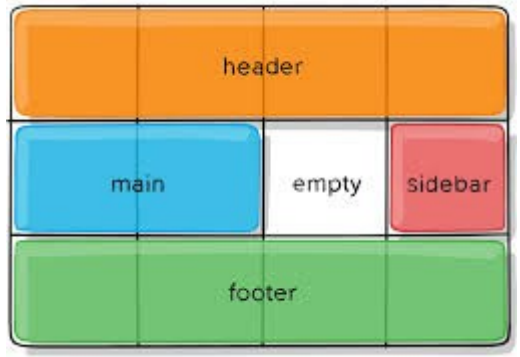
Grid (Opțional)

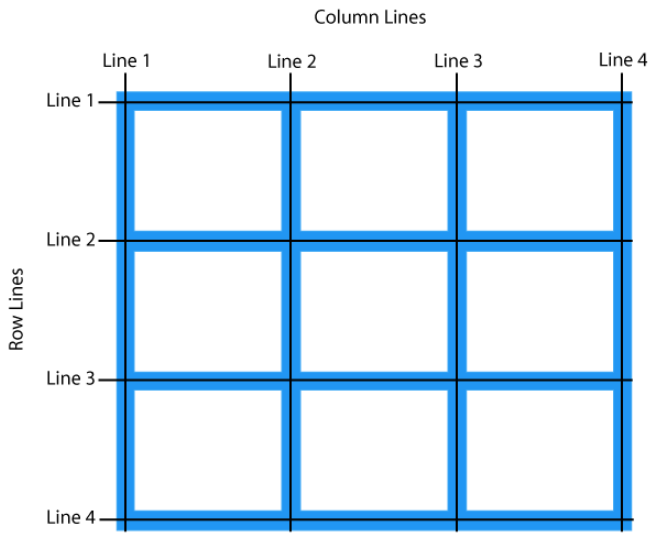
Sistem de layout utilizat pentru alinierea conținutului bazat pe un **model 2D** (două axe principale)

- Permite poziționarea elementelor copil într-o structură matriceală
- Proprietăți la nivelul container-ului părinte:
  - **display: grid;**
  - grid-template-columns, grid-template-rows
  - grid-column/row-gap
- Proprietăți la nivelul elementelor copil:
  - grid-column-start, grid-column-end
  - grid-row-start, grid-row-end

Resurse:

- https://css-tricks.com/snippets/css/complete-guide-grid/
- https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout





Media Queries (Opțional)

Sintaxă specială introdusă în CSS3 care permite aplicarea unor reguli doar în anumite condiții:

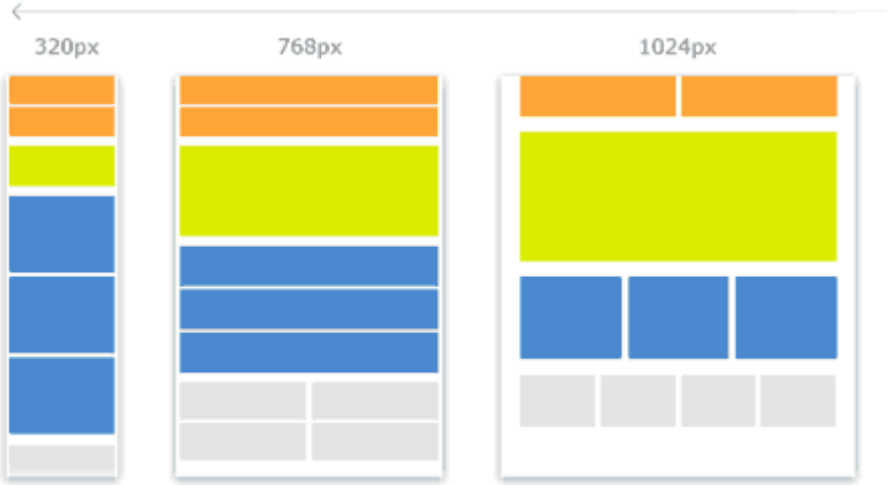
- În funcție de mediu: *all / screen / print / speech*
- În funcție de caracteristicile mediului (*width, height, ...*)

Sintaxa:

```
@media not|only mediatype and|not|only (media feature) {
  .selector { … }
}
```

Resurse:

- https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries



Limbajul JavaScript

Caracteristici

- Dinamic
- Limbaj funcțional
- Bazat pe obiecte

Utilizare

- Manipulare noduri DOM
- Evenimente
- Procesare
- Comunicare la distanță

Modalități de includere în HTML

Prin intermediul tag-ului **<script>**

A) Fișier extern

<script type="text/javascript" src="lib/test.js"></script>

B) În cadrul paginii

<script type="text/javascript">

// cod JavaScript .....

</script>

Execuție directă în fereastra de consolă

Afișare mesaje: *console.log(valori) //non-standard*

Tipuri de date

Tipuri de date de bază

- **Number**: -3.14, 6, 2
- **Boolean**: *true*, *false*
- **String**: “test”
- ***null, undefined***
- **Object**: { nume: “Ana”, varsta:7 } – referință
- **Symbol**

Obiecte speciale

- Function: *function* f() {…}
- Array: [1, 2, “trei”]

Variabile și expresii

Declarare

- Prin atribuire valoare (**nerecomandat**): a = 8; a = *false*;
- Folosind:
  - ***var***: *var* a = 3; *var* b; // function scope
  - **let**: let b = “test”; let b; // block scope
  - **const**: const b = “test”

Exemple:

a = 10; let b = “test”; b = 1.5;

const c = 7; const v = [1, 2, 3]; v[0] = 5;

Variabile și expresii

Scope

- Global sau local
- Function vs block based

Evaluare expresii

- Operatori similari cu C# (în plus === - egalitate strictă)

Exemple:

a += 7	a === 10	a < b && b > c
a > 10	a++	a = “Ana” + “-“ + “Maria”

Utilizare *template strings*: `*Numele este* ***${prenume} ${nume}****.*`

Vectori – Obiectul *Array*

Inițializare: *var v = []*; sau *var v = [1, “Ion”]; sau var v = new Array(1, 2, 3);*

Accesare elemente*: var i = v[0]; v[1] = 23;*

Dimensiune*: v.length (read / write)*

Metode:

- push(valoare) – adăugare la sfârșit
- pop() – extrage ultimul element
- indexOf(valoare) – întoarce poziția elementului (sau -1)
- sort() – sortează vectorul
- slice(index_start, index_sfârșit) – extrage un sub-vector

Parcurgere: for / for..of

Funcții

Declarare:

- *function* suma(a,b) { *return* a + b; }
- *var* suma = *function*(a,b) { *return* a + b; }

Parametri:

- Transmiși prin valoare
- Accesibili prin *arguments*

Apel - nume_funcție / expresie**(**parametri**)**:

- *var* rezultat = suma(7,3);
- *var* test = *function(*val) { return val + 1;}**(4)**;

Obiecte funcție (.apply(), .call(), .toString() …)

Closure – Environment Model of Evaluation – vezi https://sicp.comp.nus.edu.sg/chapters/52

Arrow functions

Utilizare map / reduce / filter / find

Obiecte

Obiect = colecție de proprietăți (perechi *nume* = *valoare*)

Declarare obiecte

- Literali: var ob = { nume: “Ana”, varsta:7 }
- *new*: var ob = new Object();

Accesare proprietăți

- Citire: var v = ob.varsta; sau var v = ob[“varsta”]; *for* (v *in* ob) {...}
- Modificare / adăugare: ob.varsta = 8; ob[“varsta”] = 8; ob.clasa = 2;

Metode

- adăugare: ob.test = *function*() { console.log(“test”);}
- this: ob.afisare = function() { console.log(“Nume: ” + this.nume); }
- Apel: ob.afisare();

Constructori și moștenire

Funcțiile JavaScript pot fi utilizate pentru construirea de obiecte

- Exemplu: *function* **P**ersoana(nume) { this.nume = nume; this.varsta = 1; }
- Apel: var ob2 = ***new*** **P**ersoana(“Maria”);

prototype

- Proprietate a funcției constructor
- Definește proprietățile disponibile în toate obiectele (instanțele create)
- Exemplu: Persoana.prototype.afisare = *function*() { console.log("Nume: " + this.nume); };
- Folosit și pentru implementarea moștenirii:
  - function Student() { this.facultate = “CSIE”; }
  - Student.prototype = new Persoana(“-”);
  - *var* s = new Student(); s.afisare();

Clase

Limbajul JavaScript permite utilizarea cuvântului cheie ***class*** pentru definirea funcțiilor constructor în forma:

```
class NumeClasa {
  constructor(parametri) {
    this.proprietate1 = valoare;
    this.proprietate2 = valoare;
    ...
  }
  metoda1() { ... }
  metoda2() { ... }
}
```

Moștenirea se implementează folosind cuvântul cheie ***extends***:

```
class Derivata extends NumeClasa { /* definire metode */ }
```

DOM - Structura

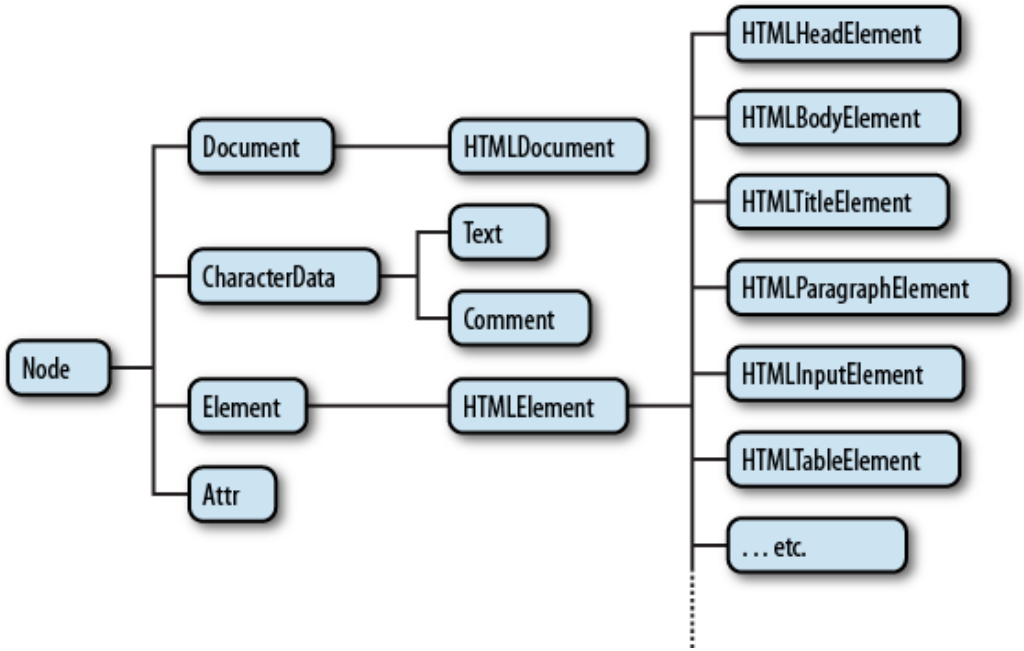
Arbore construit din obiecte JavaScript

Fiecare nod:

- Este un obiect derivat din *Node*
- Conține o colecție de referințe către nodurile copil: *childNodes / children*
- Conține referințe către nodul părinte (*parentNode*) și nodul următor (*nextSibling*)
- Conține metode pentru manipularea nodurilor copil: *appendChild*(nod), *removeChild*(nod), …

Rădăcina: *document.documentElement, document.head, document.body*

Nodurile au metode și proprietăți specifice în funcție de tag-ul HTML utilizat pentru construirea nodului



Sursa: http://web.stanford.edu/class/cs98si/slides/the-document-object-model.html

Regăsire noduri

Pe bază de identificator

- elem = *document.getElementById*(identificator)

Pe bază de selector CSS

- elem = element.*querySelector*(“selector CSS”);
- lista = element.*querySelectorAll*(“selector CSS”);

Alte metode

- lista = element.getElementsByClassName("clasa CSS");
- lista = element.getElementsByTagName("tag HTML");

Manipulare noduri

Construire nod:

- elem = *document.createElement*("tag HTML"); / document.createTextNode(“text”)
- Proprietăți: innerText, innerHTML
- Metode de adăugare nod:
  - parinte.prepend(copil)
  - parinte.append(copil)
  - nod.before(nodNou)
  - nod.after(nodNou)
  - nod.replaceWith(nodNou)
- Metode de eliminare nod:
  - parinte.removeChild(copil)
  - nod.remove()

Manipulare Atribute

Accesare atribute:

- elem.numeAtribut – accesare valoare atribut individual
- elem.attributes – colecția de atribute
- elem.hasAttribute(name) – verificare existență
- elem.getAttribute(name) – obținere valoare
- elem.setAttribute(name, value) – modificare valoare
- elem.removeAttribute(name) – ștergere atribut

Asociere date proprii:

- Atribute HTML: <tag **data**-*denumire*="valoare"></tag>
- Din JavaScript: element.**dataset**.denumire

Manipulare Noduri

Accesare atribute CSS:

- elem.style.numeAtributCSS
- elem.className – string, corespunzător atributului *class* din HTML
- elem.classList – obiect care permite manipularea listei de clase
  - (metode *add / remove / toggle / contains*)

Obținere coordonate element fata de fereastră: elem.getBoundingClientRect()

Exemplu:

```
var p = document.createElement("p");
p.innerText = "test";
document.querySelector("body").appendChild(p);
p.style.color = "red";
```

Tratare evenimente

Adăugare event handler:

- Prin atribute HTML:
  - <element atributEveniment="nume funcție">…</element>
  - Exemplu: *<a **onclick="test()"**>test*
- Prin proprietăți nod:
  - element.proprietateEveniment = funcție;
  - Exemplu: document.getElementById("test").**onclick** = function() {console.log("un mesaj");}
- **Prin metoda addEventListener:**
  - element.*addEventListener*(tip, funcție);
  - Exemplu: document.getElementById("test"). addEventListener("click", function() {console.log("un mesaj");});

Eliminare event handler:

- element.*removeEventListener*(tip, funcție);

Tratare evenimente

Accesare element sursă – *this*

Parametri eveniment – obiect *event*

- General: *target, currentTarget, type, preventDefault()*
- Tastatură: key, keyCode, *altKey, ctrlKey, shiftKey*
- Mouse: *pageX, pageY, button, altKey, ctrlKey, shiftKey*

Evenimente

- General: *load (window), DOMContentLoaded (document)*
- Tastatură*: keydown, keypress, keyup*
- Mouse*: mouseenter, mouseleave, mousemove, mouseup, mousedown, click, dblclick*

Programare execuție funcție

A) Execuție o singură dată după un interval de timp specificat

- Programare pornire:

  var id = setTimeout(funcție, durata[, param1, param2, ...]);

- Oprire:

  clearTimeout(id);

B) Execuție repetată la un interval de timp fix

- Programare pornire:

  var id = setInterval(funcție, durata[, param1, param2, ...]);

- Oprire:

  clearInterval(id);

Observație: duratele sunt exprimate în milisecunde

Comunicarea cu serverul

Prin obiecte de tip ***XMLHttpRequest***

Permite crearea, transmiterea și recepția de cereri HTTP(S)

Construire obiect cerere:

var cerere = new XMLHttpRequest();

Funcții:

- *open(method, url)* – inițializează un obiect *XMLHttpRequest*
- *send(data)* – începe procesarea asincronă a cererii
- *abort()* – anulează o cerere în desfășurare
- *setRequestHeader(header, value)* – permite transmiterea de valori în header-ul cererii
- *getResponseHeader(header)* – permite citirea valorilor din header-ele primite de la server

Comunicarea cu serverul

Proprietăți:

- *readyState* – intreg care indică starea curentă a cererii (0 – netrimisă, ..., 4 – done)
- *response* – răspunsul primit de la server de tipul indicat de *responseType*
- *status, statusText – codul HTTP pentru răspuns în format numeric și text*

Evenimente - addEventListener:

- load – cererea a fost executată și a fost primit răspunsul
- readystatechange – s-a modificat starea curentă a obiectului
- abort – cererea a fost anulată de client
- timeout – a expirat timpul alocat
- error – cererea s-a încheiat cu o eroare

Biblioteca jQuery

Colecție de funcții JavaScript pentru:

- Regăsire elemente din DOM
- Manipulare elemente, atribute și proprietăți CSS
- Înlănțuire operații pe seturi elemente
- Animație
- Comunicare HTTP

Funcția jQuery / $

1. Regăsire elemente DOM: ***$("selector CSS")***
   - *var leg =* ***$("a.test")****;*
2. Construire obiect jQuery: **$(element)**
   - *var a = document.getElementById("leg1");*
   - *var aJQuery = $(a);*
3. Construire elemente noi: ***$("cod HTML")***
   - *var p = $(" paragraf nou ");*
4. Execuție cod după inițializarea DOM: ***$(funcție)***
   - *$(function(){*
     - *// operații care utilizează DOM*
   - *})*

Obiecte jQuery

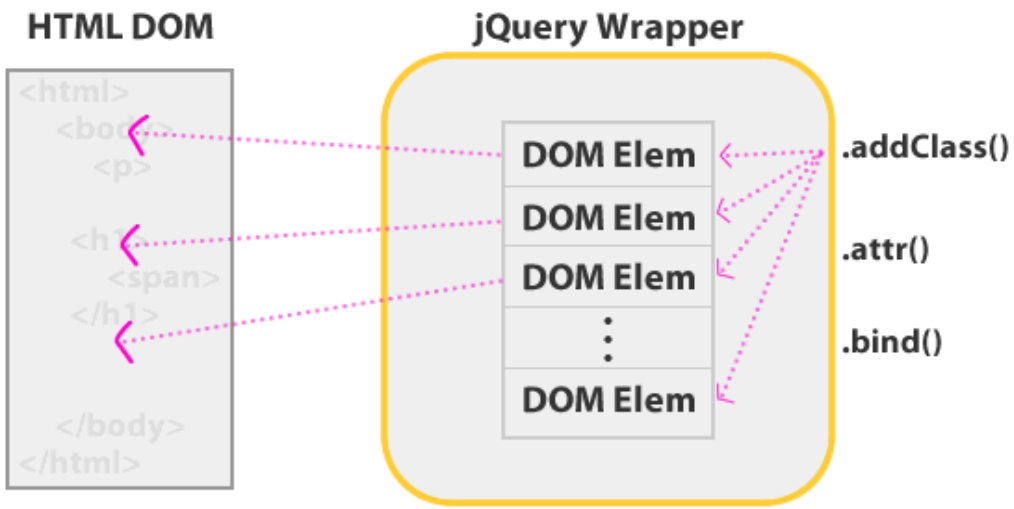
Fiecare obiect jQuery conține o colecție de 0+ referințe la obiectele DOM selectate.

Accesarea elementelor DOM:

- Operator *[]* sau funcția *get*:
  - $(“td”)[3] sau $(“td”).get(3)
  - Rezultatul obținut este un obiect DOM
- Proprietate *length*:
  - $(“td”).length

Operații asupra obiectului jQuery:

- În general se aplică pe toate elementele și întorc o referință la obiectul jQuery (pentru înlănțuire)
- Funcțiile care citesc o valoare se aplică asupra primului element și întorc valoarea citită



Exemple de operații

Citire / modificare atribute HTML

- $("#idTest").attr("href", "doc.html");
- var url = $("#idTest").attr("href");
- .text() / .val()

Citire / modificare proprietăți CSS

- $(“#idTest”).css(“color”, “red”);

Manipulare noduri DOM

- $(“#idTest”).append($(“ test ”));
- părinte.*empty()* / copil.*appendTo*(părinte) / copil.*remove*()

Tratare evenimente

- $(“#idTest”).click(func);
- $().on(“ev”, func);

Comunicarea la distanță

Obținerea de date de la server prin comenzi **GET**:

$.get(url, func) sau **$.getJSON(url, func)**

```
$.get("date.txt", function(txt) { console.log(txt); });
$.getJSON("date.txt", function(obj) { console.log(obj.proprietate); });
```

Trimiterea de date către server prin comenzi **POST**:

$.post(url, date[, func])

```
$.post(“procesare.aspx", JSON.stringify(obj), function(obj) { console.log(obj); });
```

Opțiuni complete: **$.ajax**(...)

JavaScript – Execuție asincronă (opțional)

Obiecte ***Promise*** – încapsulează o operație asincronă și callback-urile asociate

Instrucțiuni dedicate pentru obiecte *Promise*:

- ***async function***: declară o funcție asincronă care întoarce un obiect Promise
  - permite utilizarea operatorului *await* în corpul funcției
- ***await***: operator care permite execuția unei funcții asincrone ce întoarce un obiect *Promise* și continuarea execuției doar la primirea rezultatului

Exemplu:

```
async function app() {
   var raspuns = await fetch('./dateEU.json');
   var json = await raspuns.json();
   console.log(json);
}
app();
```

Resurse:

- http://javascript.info/async

II. Imagine

1. Modele de culoare

2. Grafică raster

3. Grafică vectorială

4. Animație

1. Modele de culoare

Model de culoare

- Model matematic care descrie modalitatea de reprezentare a culorilor sub formă de tupluri numerice
- Exemple: RGB, CMY

Spațiu de culori

- Modelul de culoare împreună cu instrucțiunile de reprezentare fizică
- Exemple: sRGB, AdobeRGB, Pantone

Caracteristici

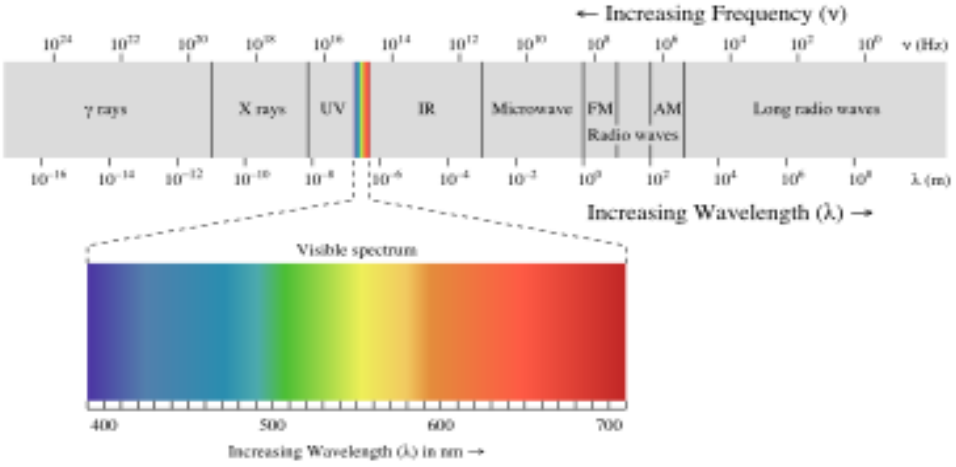
- Țin seama de modalitatea de percepere a luminii de către ochiul uman
- Culori de bază albastru (S), verde (M) și roșu (L)

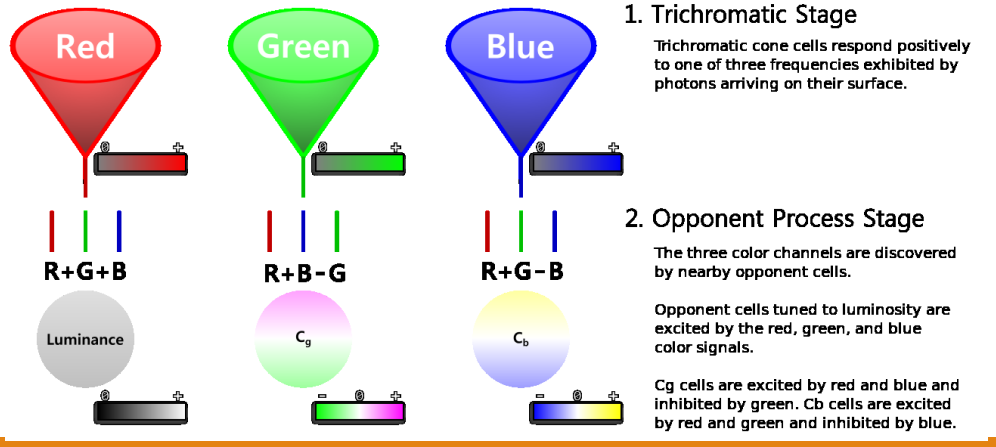
Lumina

Radiație electromagnetică cu lungimea de undă aproximativ între 400 și 700 nm

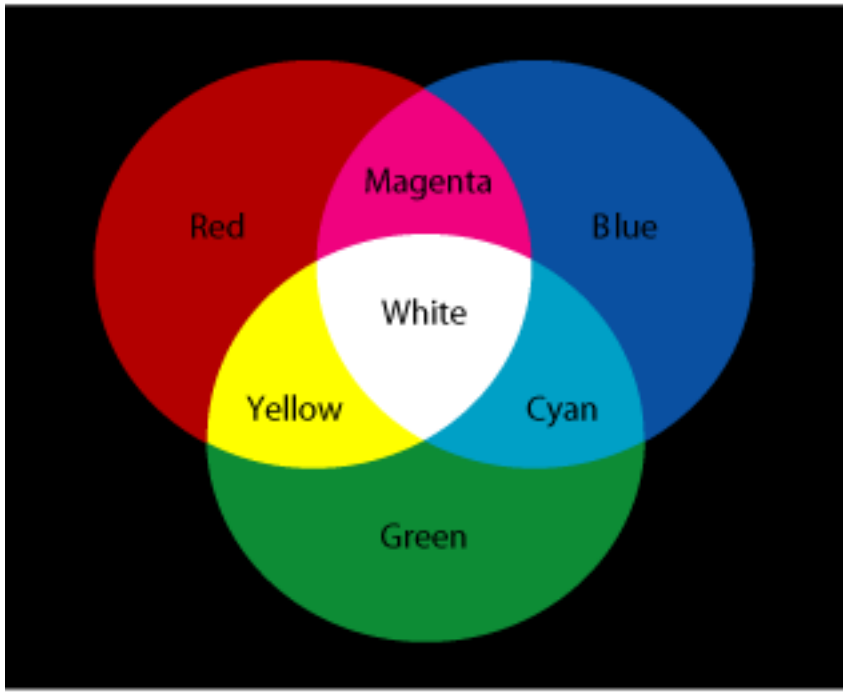
Vederea umană:

- Trei tipuri de celule conice fotosensibile (pentru culorile roșu, verde, albastru)
- Culoarea este procesată pe baza a trei componente (luminanță, C_G și C_B)

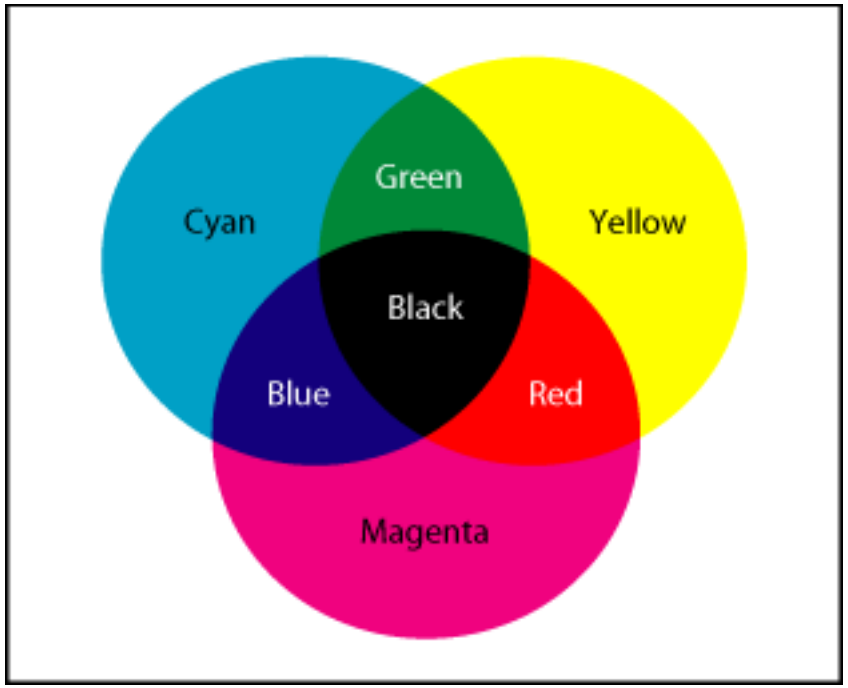




Tipuri de modele



Model aditiv



Model substractiv

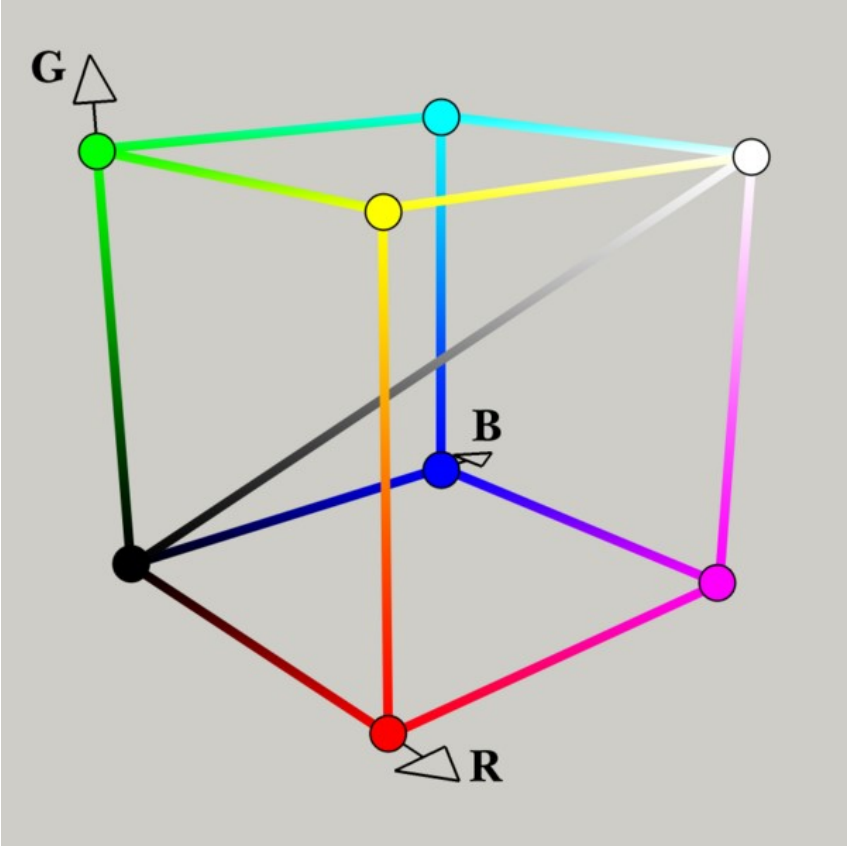
Modelul RGB

Model aditiv

Bazat pe culorile de bază roșu, verde și albastru

Culoarea variază în funcție de dispozitiv

(în lipsa unui spațiu de culoare)



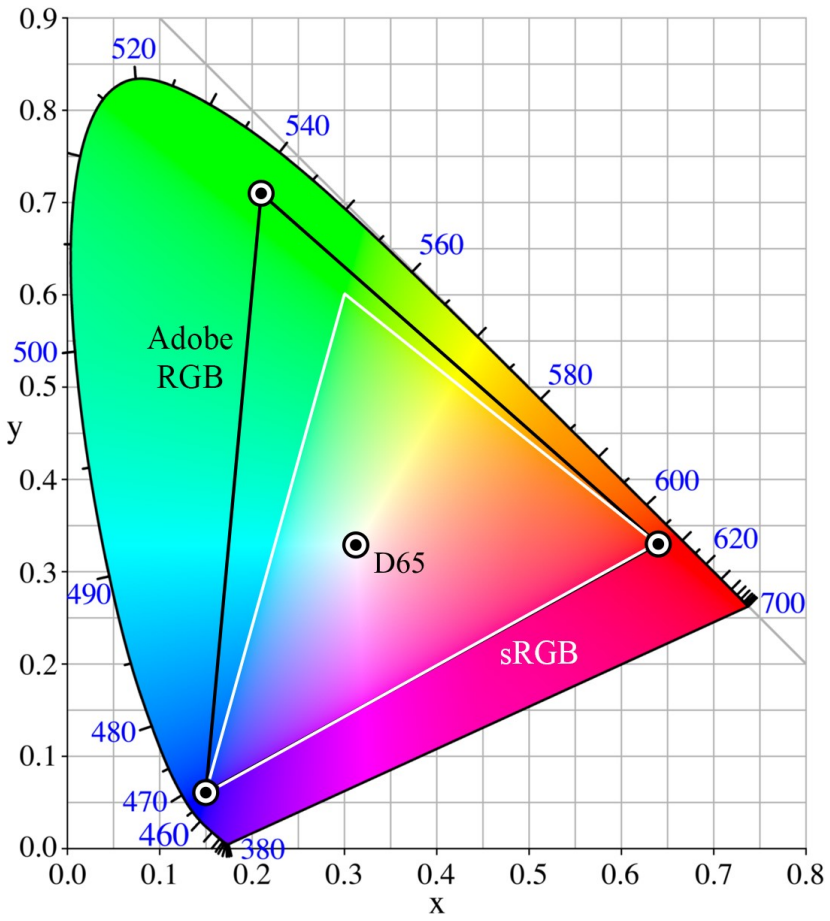
Spații de culoare RGB

sRGB

- Dezvoltat de HP și Microsoft
- Standard pentru monitoare / imprimante / web
- Utilizat ca standard implicit

AdobeRGB

- Dezvoltat de Adobe
- Acoperă aproape complet spațiul de culori CMYK



Modelul HSL (B,V)

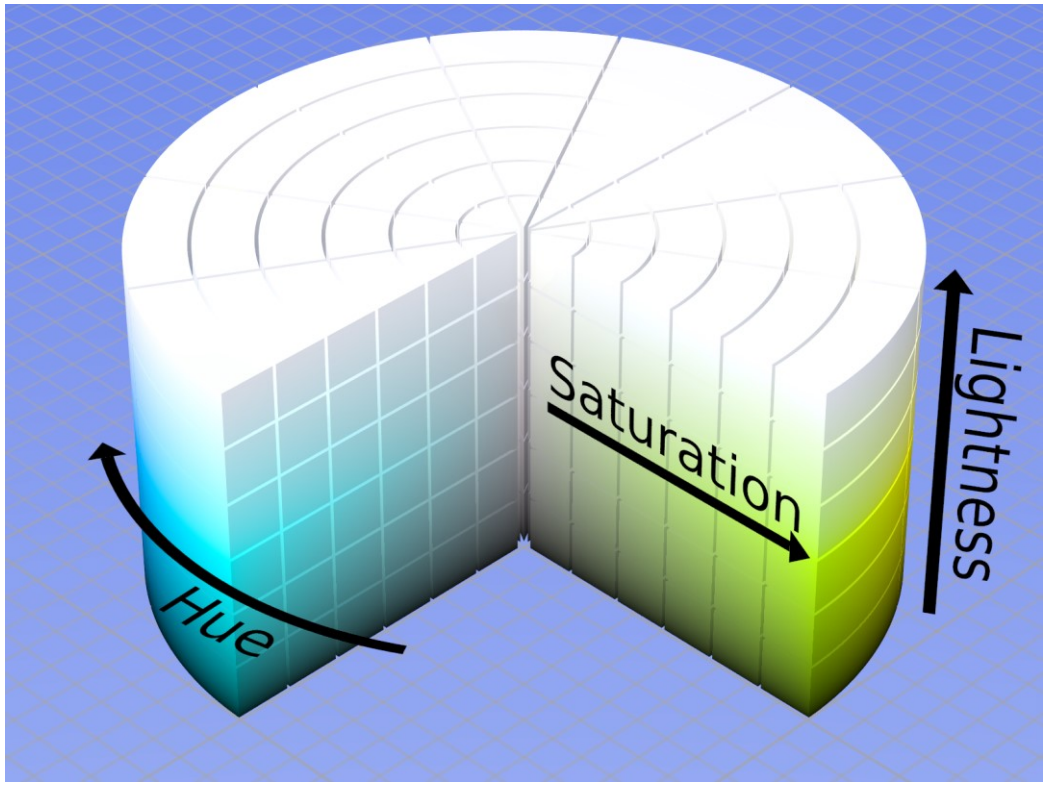
Reprezentare sub formă de coordonate cilindrice

Bazat pe aceleași culori de bază:

- Red - 0^0
- Green - 120^0
- Blue - 240^0

Axa centrală – tonuri de gri

Utilizat în special pentru selecție de culoare



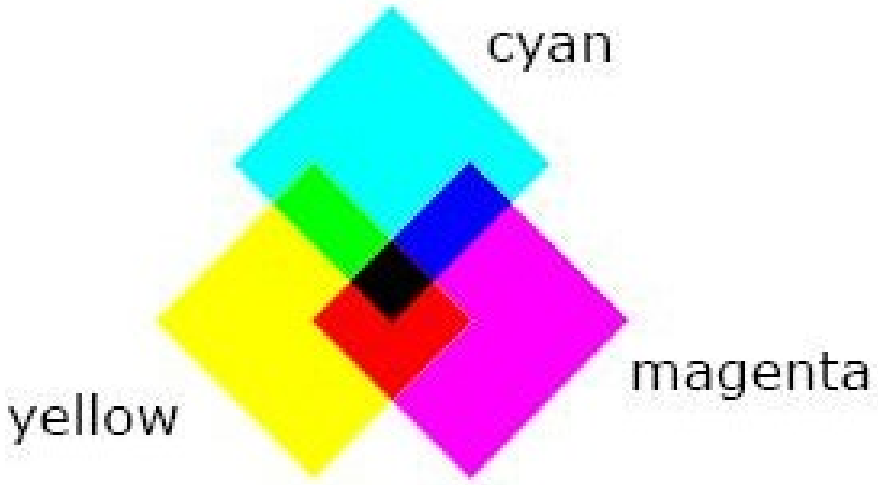
Modelul CMY(K)

Model substractiv

Culori utilizate: cyan, magenta, yellow

Maschează culorile pe o suprafață albă

Utilizare: materiale tipărite



CMY(K) model

CSS – Specificarea colorilor

Format hexazecimal:

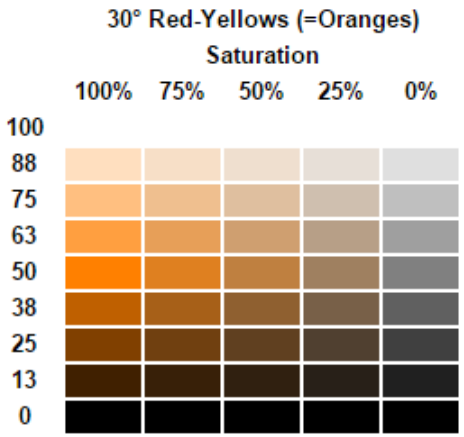
- ***#rgb*** sau ***#rrggbb***
- ***r,g,b*** sunt cifre în baza 16

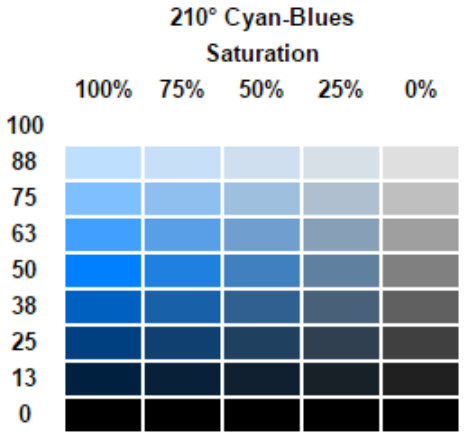
Format RGB

- ***rgb(red, green, blue)*** sau ***rgba(red, green, blue, alpha)***
- ***red, green, blue*** – numere de la 0 la 255 sau procente
- ***alpha*** – număr între 0 – transparent și 1 – opac sau procent

Format HSL

- ***hsl(hue, saturation, lightness)*** sau ***hsla(hue, saturation, lightness, alpha)***
- ***hue*** – număr de la 0 la 360
- ***saturation, lightness*** – procente
- ***alpha*** – număr între 0 – transparent și 1 – opac sau procent





Palete de culori

Tabel de corespondență

index – tuplu (R, G, B)

Utilizare

- Reducerea cantității de informație necesară pentru reprezentarea culorilor
- Pretabilă pentru imagini cu un număr redus de culori (exemplu: diagrame fără gradienți)

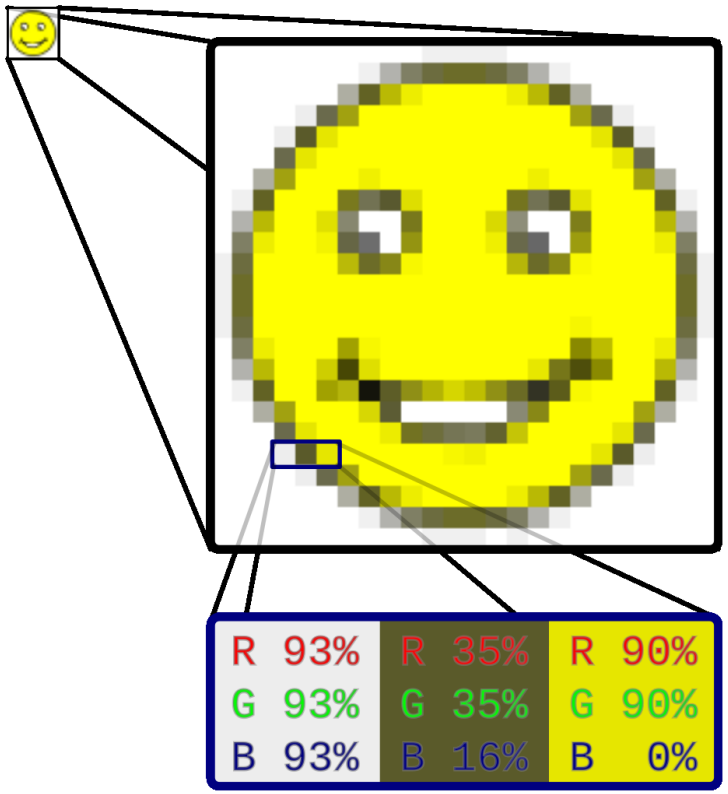
2. Grafică raster

Tipuri principale de grafică:

- **raster** și **vectorial**

Grafica raster (bitmap, matriceală)

- Reprezentare sub formă de matrice de puncte
- Fiecare punct (denumit pixel) stochează informația de culoare
- Culorile sunt stocate conform unui model de culoare
  - direct
  - prin intermediul unei palete de culori



Sursa: http://commons.wikimedia.org/wiki/File:Rgb-raster-image.svg

Informații generale

Caracteristici principale imagine raster

- Rezoluție (numărul de linii și coloane stocate în matrice, numărul total de pixeli sau densitate)
- Adâncime de culoare (cantitatea de informație stocată de către fiecare pixel)

Utilizare

- Reprezentare imagine pe monitor
- Captare imagini din surse externe

Avantaje și dezavantaje

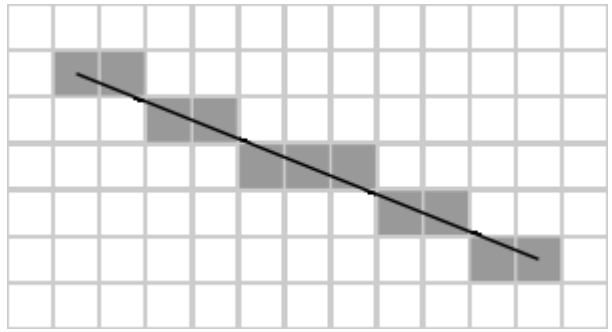
+ Poate reprezenta orice imagine
– Codaj sărac în informație (nu ia în considerare semantica imaginii)
– Dimensiune mare
– Nu se pot adapta unei scări variabile de vizualizare

Denenare elemente grafice

Presupune colorarea celulelor matricei plecând de la ecuația matematică

Exemplu: segment de dreaptă $(x_1, y_1) \rightarrow (x_2, y_2)$

Se pleacă de la ecuația dreptei $y=mx+b$ și se determină valorile pentru $m=(y_2-y_1)/(x_2-x_1)$ și $b=(y_1 - mx_1)$



Se colorează punctele corespunzătoare ecuației:

```
dx = x2 - x1; dy = y2 - y1;
for x from x1 to x2 {
    y = y1 + dy * (x - x1) / dx
    plot(x, y) }
```

(se presupun $d_x >= d_y$ și $x_1 > x_2$)

Imaginea raster în context Web

Afișarea imaginilor în HTML: <**img** alt="…" src="…">

- Utilizare nod DOM (HTMLImageElement)
  - *width*, *height*: dimensiunea obiectului în fereastră
  - *naturalWidth*, *naturalHeight*: dimensiunea matricei raster
  - *src*: URL-ul imaginii sursă; modificarea determină începutul procesului de încărcare (asincron)
  - eveniment *load*
- Exemplu:

```
var image = $("<img>")
        .load(function () { $("body").append($(this)); })
        .attr("src", "media/Penguins.jpg");
```

Elementul **canvas**

HTML – includere în document:

```
<canvas id= "test" width= "250" height="150"></canvas>
```

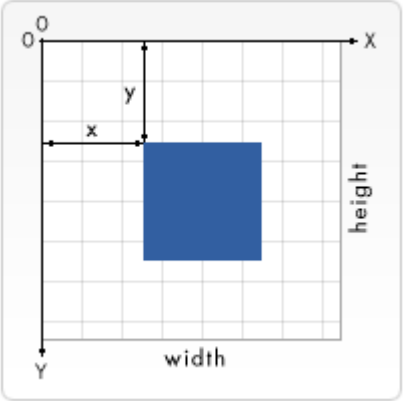
JavaScript – obținere referință context grafic:

```
// obținere obiect HTMLCanvasElement din DOM
var canvas = document.getElementById('test');  // sau var canvas = $('test')[0];

var w = canvas.width, h = canvas.height;

// obținere context grafic (obiect de tip CanvasRenderingContext2D)

var ctx = canvas.getContext('2d');
```

canvas – Modalități desenare

Desenare directă – dreptunghiuri:

- *fillRect(x, y, width, height)* – umplere suprafață
- *strokeRect(x, y, width, height)* – desenare contur
- *clearRect(x, y, width, height)* – ștergere suprafață



Desenare folosind căi:

- ***beginPath()*** – deschide o cale nouă
- instrucțiuni de desenare / poziționare
- opțional *closePath()* – închide calea (unește cu punctul de început)
- ***fill()*** – umple calea și / sau ***stroke()*** – desenează liniile

canvas – Instrucțiuni desenare

Deplasare:

moveTo(x,y) – modifică poziția curentă

Desenare:

lineTo(x, y) – adaugă o linie de la poziția curentă la punctul specificat

rect(x, y, width, height) – adaugă un dreptunghi

arc(x, y, radius, startAngle, endAngle, anticlockwise) – adaugă un arc de cerc

quadraticCurveTo(cp1x, cp1y, x, y) – adaugă o curbă cvadratică (un punct de control)

bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y) – adaugă o curbă Bézier (două puncte de control)

Desenare text: *fillText* sau *strokeText(string, x, y)*

canvas – Atribute

Salvare și restaurare atribute context folosind stiva: ***save()*** și ***restore()***

Atribute:

- *fillStyle* sau *strokeStyle* = “culoare” – modifică culoarea de desenare
  - *ctx.fillStyle = "#FFA500";*
  - *ctx.strokeStyle = "rgba(255,165,0,1)";*
- *lineWidth* = dimensiune – modifică grosimea liniilor
- *lineCap, lineJoin, miterLimit* – modifică proprietățile liniilor (rotunjire capete, îmbinări)
- *font* = “specificație font” – modifică caracteristicile textului (exemplu: "bold 18px Arial")
- *textAlign* – poziția textului față de coordonata x *(left, right* sau *center)*
- *textBaseline* – poziția textului față de coordonata y (*top, hanging, middle, alphabetic* sau *bottom*)

canvas – Desenare curbe

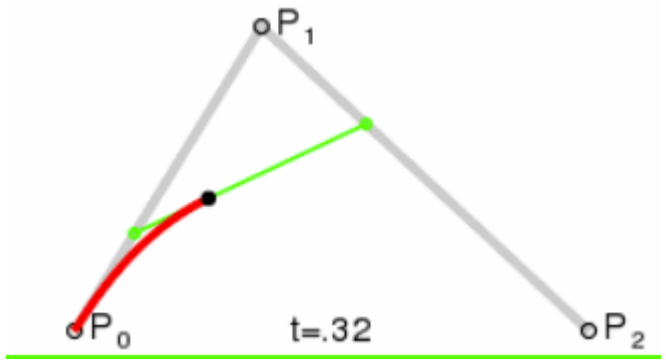
Suport pentru desenare curbe *Bézier*

- cuadratice (un punct de control)
- cubice (două puncte de control)

Formule:

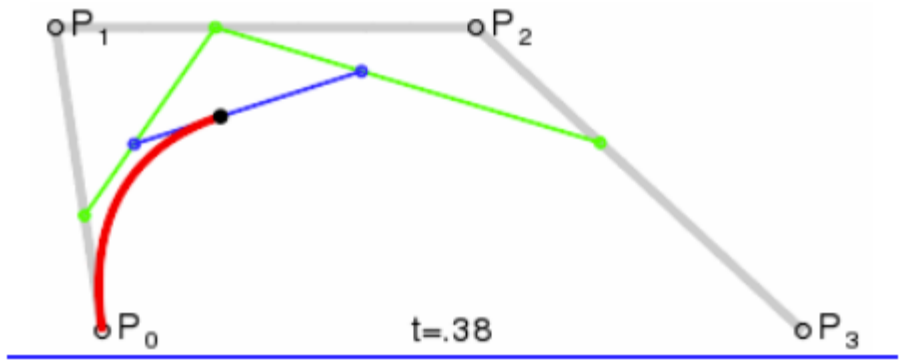
$$P = (1-t)P_1 + tP_2$$

$$P = (1-t)^2P_1 + 2(1-t)P_2 + t^2P_3$$

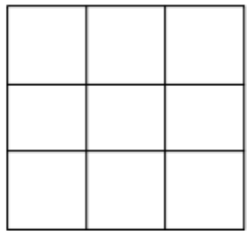


Algoritmul *de Casteljau* (cuadratice):

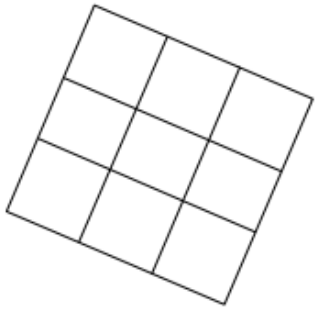
- se consideră punctele P_0, P_1 și P_2
- se construiesc segmentele P_0P_1 și P_1P_2
- Pentru t de la 0 la 1
  - Se determină punctul aflat la o distanță proporțională t față de începutul segmentului pentru P_0P_1 și P_1P_2
  - Se construiește un segment nou din cele două puncte obținute și se repetă procesul



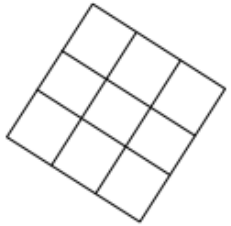
Tipuri de transformări 2D



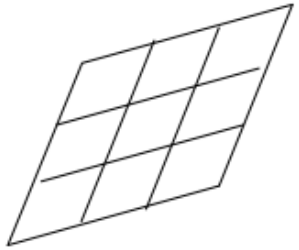
Identică



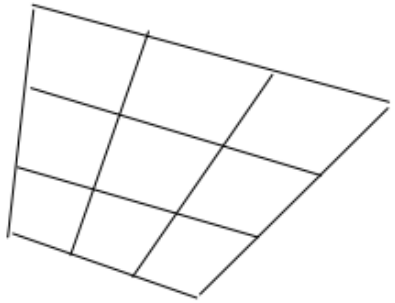
Euclidiană
(translație, rotație)



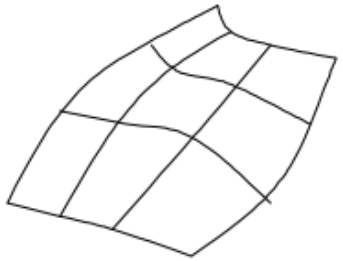
Helmert
(+scalare)



Afină
(+deformare)



Proiectivă
(+paralelism)

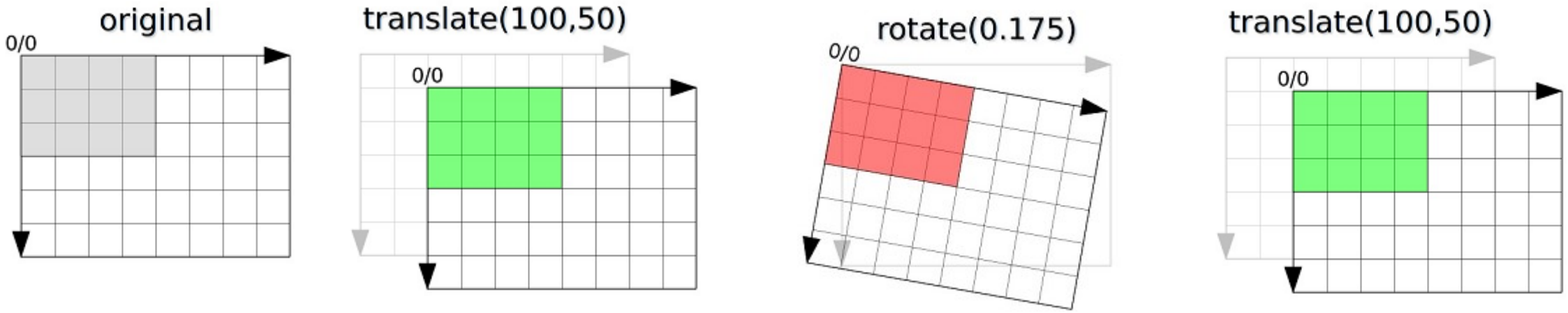


Polinomială
(+proprietăți geometrice)

canvas – Transformări

Operații de transformare simplă:

- *translate(x, y)* – translatează sistemul de coordonate cu numărul de pixeli specificați
- *rotate(angle)* – rotește sistemul de coordonate cu unghiul specificat (în radiani)
- *scale(x, y)* – scalează sistemul de coordonate cu factorii specificați (în [0…1])



canvas - Transformări

Forma generală pentru transformări afine:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{21} & d_x \\ m_{12} & m_{22} & d_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Compunere transformare (prin înmulțire):

transform(m11, m12, m21, m22, dx, dy)

Înlocuire transformare curentă:

setTransform(m11, m12, m21, m22, dx, dy)

resetTransform() – revenire la sistemul de coordonate standard

canvas - Transformări

Semnificație parametri:

- m_{11}: scalare pentru axa *Ox*
- m_{12}: rotire pentru axa *Ox*
- m_{21}: rotire pentru axa *Oy*
- m_{22}: scalare pentru axa *Oy*
- d_x: translatare pentru axa *Ox*
- d_y: translatare pentru axa *Oy*

canvas – Desenare imagini

Desenare fără scalare:

drawImage(image, x, y)

Desenare cu preluare porțiune imagine si scalare:

drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)

Surse posibile:

- element **img**
- alt element **canvas**
- element **video**

canvas – Acces raster

Stocare raster – obiect *ImageData*:

- *width, height*: dimensiunile matricei
- *data*: vector care conține matricea liniarizată

Operații:

- *getImageData(left, top, width, height)*: extrage o porțiune din imagine ca obiect *ImageData*
- *putImageData(imageData, x, y)*: aplică datele pe imaginea afișată în *canvas*

Restricții de origine

canvas – Access raster

Parcurgere elemente:

```
var imageData = context.getImageData(
                    0, 0, canvas.width, canvas.height);

for (var y = 0; y < canvas.height; y++) {
    for (var x = 0; x < canvas.width; x++) {
        var i = (y * canvas.width * 4) + x * 4;

        var rosu = imageData.data[i];                // [0..255]
        var verde = imageData.data[i+1];             // [0..255]
        var albastru = imageData.data[i+2];          // [0..255]
        var transparenta = imageData.data[i+3];      // [0..255]
    }
}
```

Efecte simple de culoare

Filtrare culoare

- Exemplu pentru roșu: **r’ = r; g = 0; b = 0**

Negativ

- **r‘ = 255 – r; g‘ = 255 – g; b‘ = 255 – b;**

Transformare în tonuri de gri

- **r‘ = g’ = b’ = 0.299 * r + 0.587 * g + 0.114 * b**

Modificare strălucire

- **r‘ = r + *valoare*; if (r > 255) r = 255 else if (r < 0) r = 0;**
- similar pentru celelalte două componente

Modificare contrast

- **r' = (((r / 255) - 0.5) * *valoare* + 0.5) * 255; if (r > 255) r = 255 else if (r < 0) r = 0;**
- similar pentru celelalte două componente

Transformare RGB → HSL

$$M = \max(R, G, B)$$
$$m = \min(R, G, B)$$
$$C = M - m$$

$$H' = \begin{cases} \text{undefined}, & \text{if } C = 0 \\ \frac{G-B}{C} \bmod 6, & \text{if } M = R \\ \frac{B-R}{C} + 2, & \text{if } M = G \\ \frac{R-G}{C} + 4, & \text{if } M = B \end{cases}$$
$$H = 60^\circ \times H'$$

$$L = \frac{1}{2}(M + m) = 0.5 * max(R, G, B) + 0.5 * min(R, G, B)$$

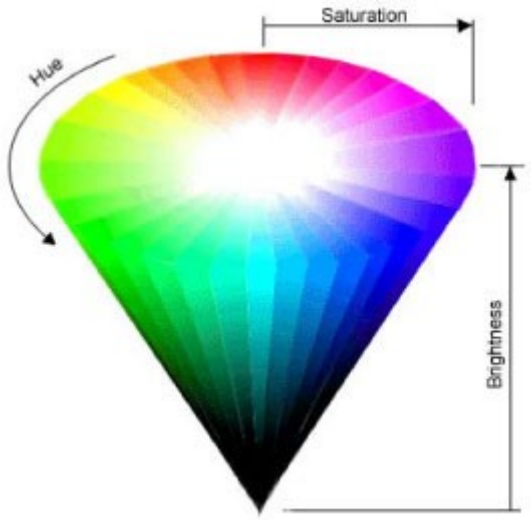
$$S_{HSL} = \begin{cases} 0, & \text{if } L \in \{0, 1\} \\ \frac{C}{1-|2L-1|}, & \text{otherwise} \end{cases}$$

Transformare HSL → RGB

$$C = (1 - |2L - 1|) \times S_{HSL}$$
$$H' = \frac{H}{60^\circ}$$
$$X = C(1 - |H' \bmod 2 - 1|)$$

$$(R_1, G_1, B_1) = \begin{cases} (0,0,0) & \text{if } H \text{ is undefined} \\ (C,X,0) & \text{if } 0 \le H' < 1 \\ (X,C,0) & \text{if } 1 \le H' < 2 \\ (0,C,X) & \text{if } 2 \le H' < 3 \\ (0,X,C) & \text{if } 3 \le H' < 4 \\ (X,0,C) & \text{if } 4 \le H' < 5 \\ (C,0,X) & \text{if } 5 \le H' < 6 \end{cases}$$

$$m = L - \tfrac{1}{2}C$$
$$(R,G,B) = (R_1 + m, G_1 + m, B_1 + m)$$



Histograma unei imagini

Permite analiza distribuției tonurilor în cadrul unei imagini

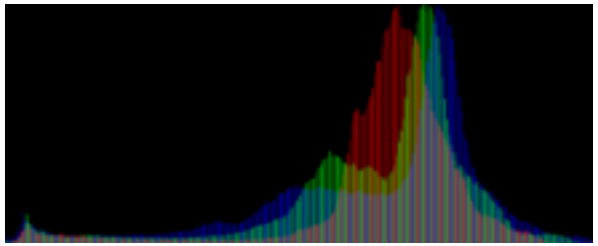
Construcție:

- O***x***: intensitatea (de obicei 0-255)
- O***y***: numărul de pixeli existenți pentru fiecare valoare a intensității

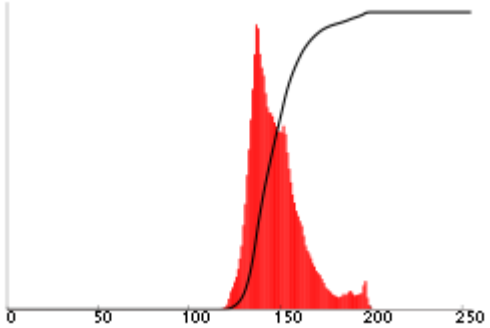
Poate fi construită pentru fiecare canal în parte

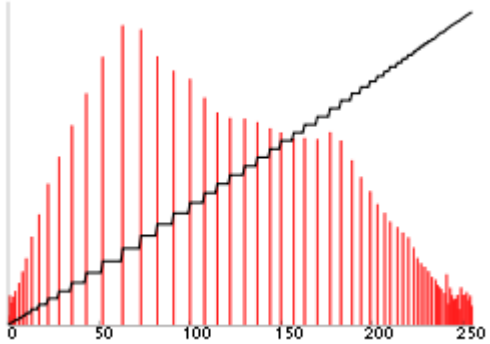
Exemple de utilizări:

- Îmbunătățire contrast prin egalizare histogramă
- Descoperire muchii / segmentare imagine



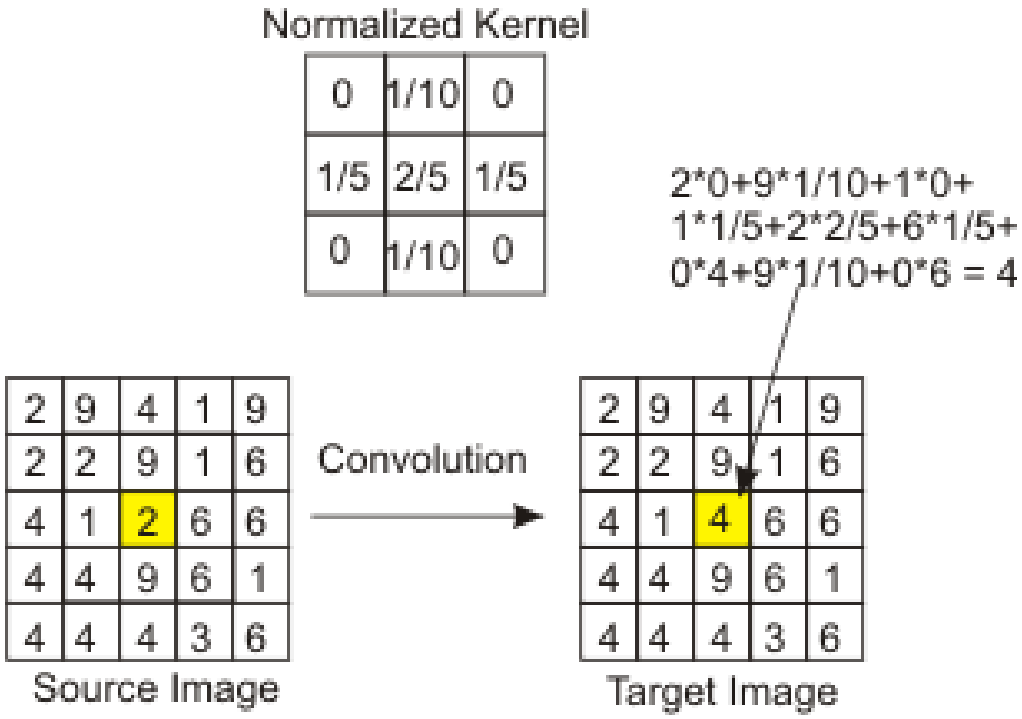
Exemplu egalizare histogramă





Filtre de convoluție

Calculează valoarea fiecărui pixel în funcție de valorile pixelilor alăturați



Filtre de convoluție

Algoritm general:

pentru fiecare valoare v(x,y) din matricea originală

acumulator = 0

pentru fiecare valoare k(i,j) din matricea de convoluție

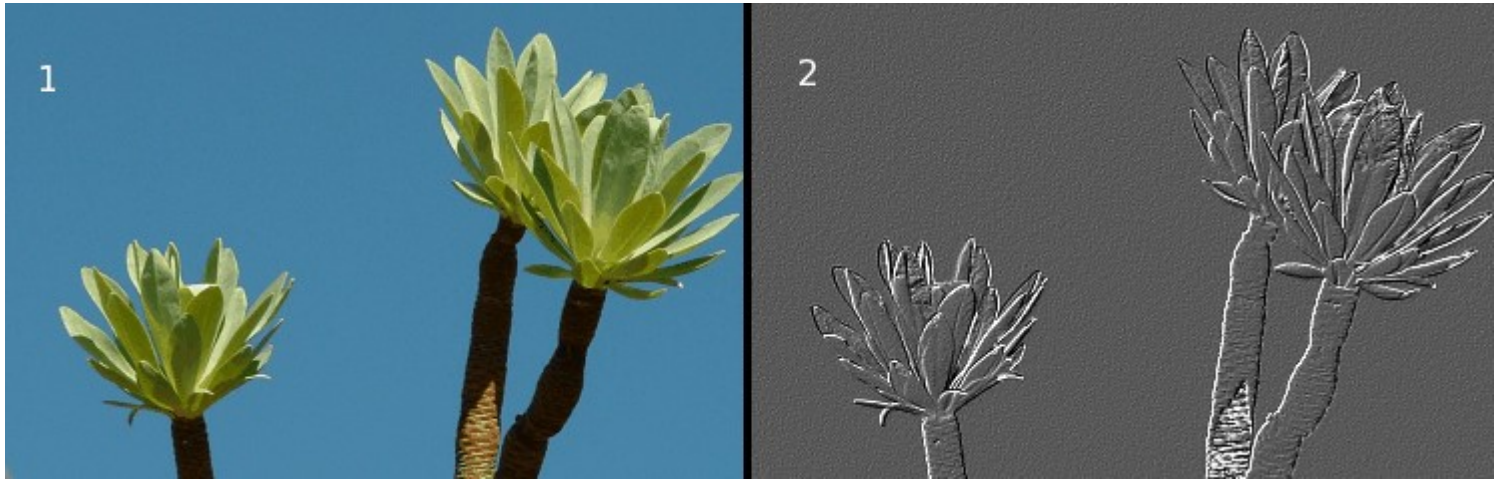
*acumulator = acumulator + v(x+i,y+j)*k(i,j)*

v'(x,y) = acumulator (trunchiat la 0..255)

Observații:

- se aplică pe fiecare canal de culoare în parte
- tratare specială pentru pixelii din margine

Filtre de convoluție – Exemplu emboss



$$\begin{pmatrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{pmatrix}_{+127}$$

Filtre de convoluție – Alte exemple

Gaussian Blur

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}_{/16}$$

Sharpen

$$\begin{pmatrix} 0 & -2 & 0 \\ -2 & 11 & -2 \\ 0 & -2 & 0 \end{pmatrix}_{/3}$$

Edge Detection

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}_{+127}$$

Formate de stocare

BMP (Microsoft Windows Bitmap)

- Formatul standard de stocare pe platforma Microsoft Windows
- Suportă date necomprimate sau comprimate folosind algoritmul RLE
- Monocromă sau în culori pe 4, 8, 16, 24 sau 32 de biţi
- Suportă palete de culori

JPEG (Joint Photographics Experts Group)

- Stocare comprimată cu pierdere de informație conform standardului JPEG
- Rate de compresie diferite selectabile de către utilizator
- Utilizat pentru imagini fotografice (cu gradații fine de culoare)
- Nu este potrivit pentru
  - text, linii sau alte imagini care prezintă un contrast foarte mare
  - Editări multiple (se pierde calitate la fiecare etapă de compresie / decompresie)

Formate de stocare

GIF (Graphics Interchange Format)

- Folosit în special pentru transferul imaginilor de maxim maxim 64K x 64K
- Pretabil pentru diagrame, text logo-uri (contrast puternic și număr limitat de culori)
- Suportă maxim 256 culori prin intermediul unei palete de culori
- Poate stoca mai multe cadre (pentru animație)
- Algoritm de compresie fără pierdere de informație Lempel-Ziv-Welch (LZW)

TIFF (Tag Image File Format)

- Format portabil și extensibil utilizat în special pentru imagini scanate
- Suportă stocarea mai multor imagini într-un singur fișier
- Suportă mai mulți algoritmi de compresie (RLE, LZW sau JPEG)

Alte formate: ICO - Icon Resource File, PNG - Portable Network Graphics, DIB - Device Independent Bitmap, PCX - PC PaintBrush File Format

Compresia **Run-length encoding** (RLE)

Secvențele de valori identice consecutive sunt înlocuite cu perechi de forma

(valoare, număr apariții)

Exemplu:

AAAAAAAAAAAAAABBBBBBBBBBAAAAAAAAAAAAABBBBBBCCC

14A10B13A6B3C

Caracteristici

- Rată mică de compresie
- Se pretează pentru imagini cu zone mari de aceeași culoare
- Utilizat în special pentru fișiere BMP cu paletă de culori

Compresia **Lempel–Ziv–Welch** (LZW)

Algoritm de compresie universal bazat pe dicționar

Descriere compresie:

1. Se construiește dicționarul inițial (toate șirurile de lungime 1)
2. Se caută cel mai lung șir W din dicționar care se potrivește cu șirul de la intrare
3. Se elimină W din șirul de intrare
4. Se adaugă W urmat de următorul caracter în dicționar
5. Se continuă cu pasul 2

Decompresie: se parcurge șirul codificat și se reconstruiește dinamic dicționarul

Variante: coduri de lungime variabilă, cod pentru reinițializare dicționar

Utilizat pentru fișiere de tip GIF

Compresia **Huffman**

Algoritm universal de compresie

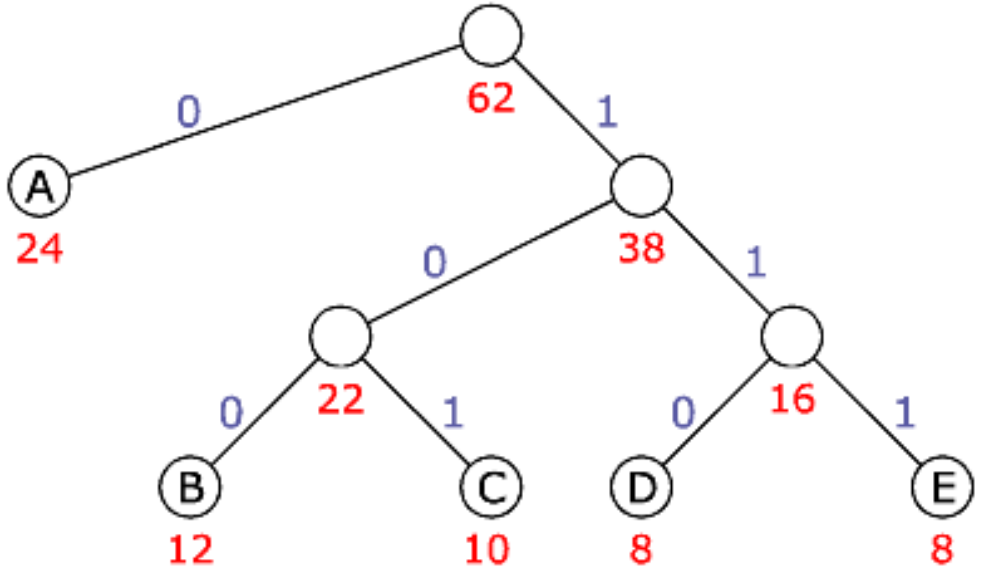
Codificare optimă de lungime variabilă pentru fiecare simbol în funcţie de frecvenţa de apariţie

Datele salvate:

- dicţionarul
- datele originale recodificate

Decodificare:

- translaţie simbol cu simbol pe baza dicţionarului salvat



Compresia **JPEG**

Compresie specializată cu pierdere de informaţie pentru imagine raster

Rezultate foarte bune pentru fotografii (variaţii fine de luminozitate şi culoare)

Tipuri de compresie:

- **secvenţial** – codaj bazat pe transformarea cosinus discretă cu blocurile procesate în ordinea apariției
- **progresiv** – codaj bazat pe transformarea cosinus discretă cu blocurile procesate prin mai multe treceri asupra imaginii
- **progresiv fără pierdere** – folosește doar algoritmi de compresie fără pierdere de informație
- **progresiv ierarhic** – codifică imaginea la rezoluții din ce în ce mai mari

Etapele compresiei **JPEG**

Etapele de **compresie** JPEG File Interchange Format (JFIF)

1. Translatarea modului de culoare din RGB în $Y'C_BC_R$

2. Reducerea rezoluţiei pentru componentele C_B şi C_R

3. Imaginea se descompune în blocuri de dimensiune 8x8 pixeli

4. Se aplică transformata cosinus discretă pe fiecare bloc în parte

5. Aplicarea matricei de cuantizare (pierdere de informație)

6. Blocurile rezultate în urma cuantizării sunt comprimate folosind RLE și Huffman

Decodificare: se aplică pașii în ordine inversă

JPEG -Transformata cosinus discretă

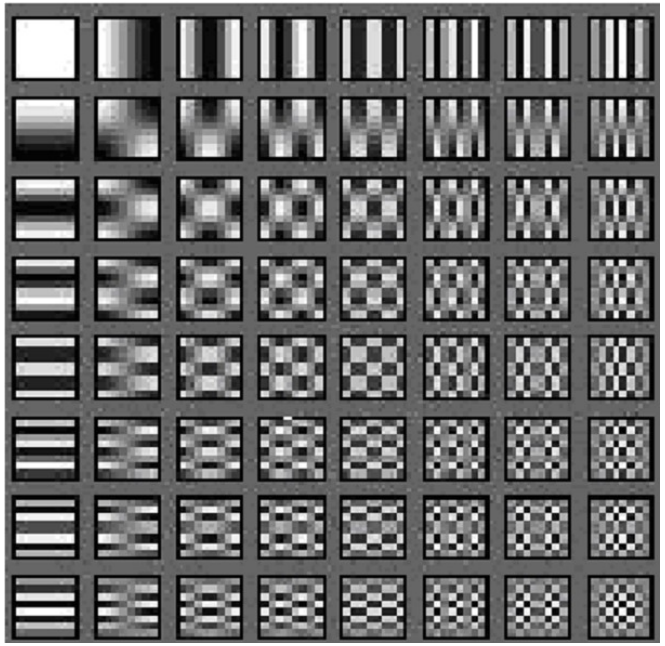
$$\text{DCT: } G_{ij} = \frac{1}{4} C_i C_j \sum_{x=0}^{7} \sum_{y=0}^{7} p_{xy} \cos\left(\frac{(2y+1)j\pi}{16}\right) \cos\left(\frac{(2x+1)i\pi}{16}\right)$$

$$\text{IDCT: } p_{xy} = \frac{1}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} C_i C_j G_{ij} \cos\left(\frac{(2y+1)j\pi}{16}\right) \cos\left(\frac{(2x+1)i\pi}{16}\right)$$

unde

$$C_i = C_j = \frac{1}{\sqrt{2}}, \quad \text{dacă } i = j = 0$$

$$C_i = C_j = 1, \quad \text{în rest}$$



JPEG -Transformata cosinus discretă

139	144	149	153	155	155	155	155
144	151	153	156	159	156	156	156
150	155	160	163	158	156	156	156
159	161	162	160	160	159	159	159
159	160	161	162	162	155	155	155
161	161	161	161	160	157	157	157
162	162	161	163	162	157	157	157
162	162	161	161	163	158	158	158

235.6	-1.0	-12.1	-5.2	2.1	-1.7	-2.7	1.3
-22.6	-17.5	-6.2	-3.2	-2.9	-0.1	0.4	-1.2
-10.9	-9.3	-1.6	1.5	0.2	-0.9	-0.6	-0.1
-7.1	-1.9	0.2	1.5	0.9	-0.1	0.0	0.3
-0.6	-0.8	1.5	1.6	-0.1	-0.7	0.6	1.3
1.8	-0.2	1.6	-0.3	-0.8	1.5	1.0	-1.0
-1.3	-0.4	-0.3	-1.5	-0.5	1.7	1.1	-0.8
-2.6	1.6	-3.8	-1.8	1.9	1.2	-0.6	-0.4

JPEG - Cuantizare

235.6	-1.0	-12.1	-5.2	2.1	-1.7	-2.7	1.3
-22.6	-17.5	-6.2	-3.2	-2.9	-0.1	0.4	-1.2
-10.9	-9.3	-1.6	1.5	0.2	-0.9	-0.6	-0.1
-7.1	-1.9	0.2	1.5	0.9	-0.1	0.0	0.3
-0.6	-0.8	1.5	1.6	-0.1	-0.7	0.6	1.3
1.8	-0.2	1.6	-0.3	-0.8	1.5	1.0	-1.0
-1.3	-0.4	-0.3	-1.5	-0.5	1.7	1.1	-0.8
-2.6	1.6	-3.8	-1.8	1.9	1.2	-0.6	-0.4

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

$$Cq(u,v)=\text{round}[C(u,v)/N(u,v)]$$

235.6/16 -> 15
-22.6/12 -> -2

15	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
-1	-1	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

3. Animație

Modificarea rapidă a imaginii vizualizate prin modificarea poziției, formei sau dimensiunii unui obiect din imagine

Stocarea numerică a animației presupune reținerea elementelor independente ce compun mișcarea în raport cu factorul timp.

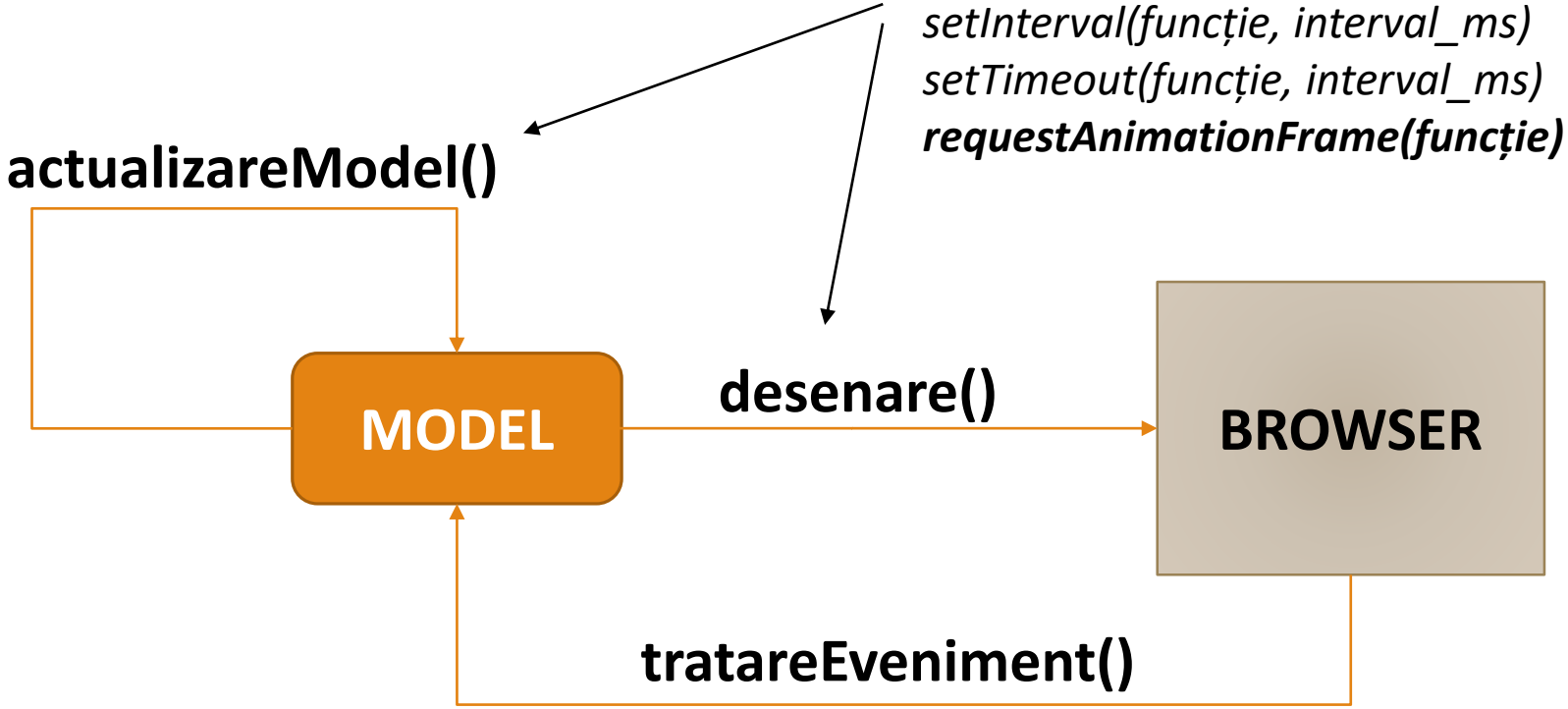
Crearea iluziei de mișcare se realizează prin afișarea rapidă de imagini statice ușor modificate

Tehnici principale:

- Tehnica filmului
- Cadre cheie
- Schimbarea culorii

Animație - JavaScript

Apelate la un anumit interval de timp prin intermediul uneia dintre funcțiile:
setInterval(funcție, interval_ms)
setTimeout(funcție, interval_ms)
requestAnimationFrame(funcție)



4. Grafică vectorială

Bazată pe descrierea matematică a obiectelor componente ale imaginii

Avantaje:

- Menține semantica – editare la nivel de obiect grafic
- Dimensiune redusă
- Independente de scara de vizualizare

Dezavantaj

- Nu poate reprezenta fidel orice fel de informație

Formate de reprezentare și stocare

SVG (Scalable Vector Graphics)

- Format generic bazat pe XML pentru reprezentări vectoriale 2D
- Suportă animație și interactivitate

DXF (Drawing Exchange Format)

- formatul vectorial lansat de firma Autodesk pentru produsul software AutoCAD

EPS (Encapsulated Post Script)

- formatul firmei Adobe pentru imagini vectoriale
- se bazează pe un limbaj de descriere numit Post Script

SHP (Shapefile)

- formatul firmei ESRI pentru descrierea datelor spaţiale de tip: punct, polilinie şi poligon
- utilizat la reprezentarea elementelor geografice în sisteme de tip GIS

SVG – Scalable Vector Graphics

Poate fi:

- inclus direct într-o pagină HTML
- controlat prin intermediul CSS și JavaScript

Exemplu:

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

SVG – Elemente de bază

Linii

<line x1="start-x" y1="start-y" x2="end-x" y2="end-y">

Dreptunghiuri

<rect x=“start-x" y=“start-y" width=“width" height=“height“ rx=“radius-x” ry=“radius-y”/>

Cercuri

<circle cx=“center-x" cy=“center-y" r=“radius"/>

SVG – Elemente de bază

Elipse

<ellipse cx=“center-x" cy=“center-y" rx=“radius-x“ ry=“radius-y”/>

Poligoane

<polygon points=“x1,y1 x2,y2 …”/>

Poli-linii

<polyline points=“x1,y1 x2,y2 …”/>

Text

<text x=“start-x" y=“start-y">conținut</text>

SVG - Atribute

Setate prin intermediul CSS

Atribute de bază

- stroke – culoare linie
- stroke-opacity – opacitate linie
- stroke-width – dimensiune linie

- fill – culoare suprafață
- fill-opacity – opacitate suprafață

SVG – Grupare și reutilizare

Grupare elemente

<g id=“id_grup”>… <!– elemente --> </g>

Definire elemente fără afișare

<defs>… <!– definire grupuri --> </defs>

Reutilizare

<use xlink:href="#id_grup" x=“30" y=“14"/>

SVG – Transformări

Se aplică pentru un element prin intermediul atributului ***transform***

Modifică întreg sistemul de coordonate

Transformări disponibile:

- scale(sx[, sy])
- translate(tx, ty)
- rotate(unghi[, cx, cy])
- skewX(unghi) / skewY(unghi)

SVG – Manipulare din JavaScript

Similar cu manipularea elementelor HTML

Particularități

- la construirea elementului trebuie specificat namespace-ul pentru SVG:
  - *document.createElementNS("http://www.w3.org/2000/svg", „TAG_SVG")*
- în jQuery se utilizează funcția *attr* pentru modificarea atributelor (în loc de .width, .height, …)

Exemplu:

```
$(document.createElementNS("http://www.w3.org/2000/svg", "rect"))
        .attr({x:160, y:160, width:12, height:12})
        .appendTo($("#desen"));
```

III. Sunet

1. Noțiuni generale
2. Numerizarea sunetului
3. Formate audio
4. Compresia sunetului
5. Sunetul în context Web

1. Noțiuni generale

Sunetul este o vibrație propagată printr-un mediu material sub forma unei unde mecanice.

- Din punct de vedere fiziologic: senzația produsă asupra organului auditiv de către vibrațiile materiale ale corpurilor și transmise pe calea undelor acustice;
- Urechea umană percepe vibrații în intervalul 20-20000Hz
- Zgomot: caz particular de sunet caracterizat prin lipsa încărcăturii informaționale

Redare / receptare sunet

- Prin intermediul difuzorului și microfonului
- Ambele fac conversie semnal electric – vibrație mecanică
- Folosesc principiile inducției electromagnetice

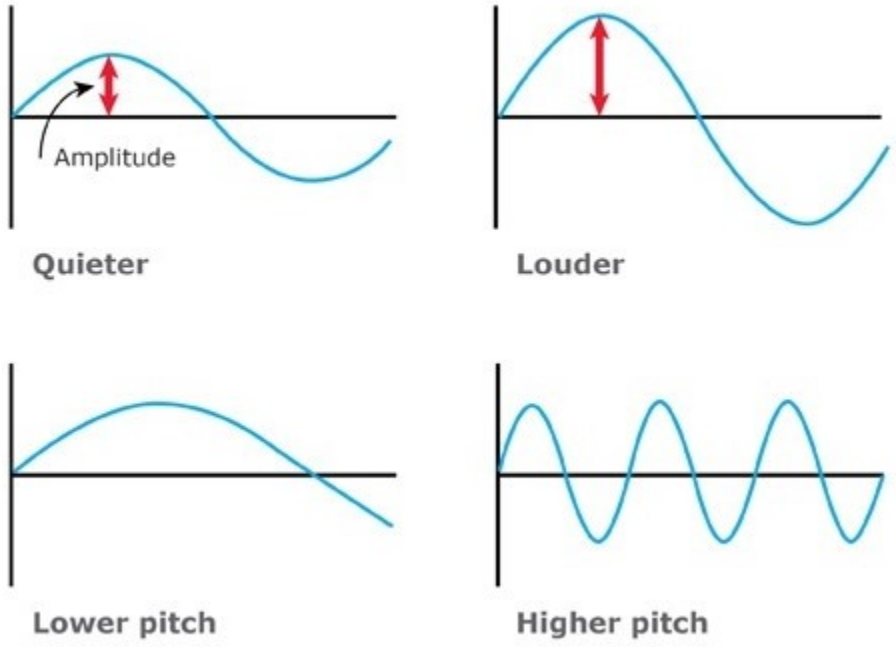
Reprezentare și caracteristici

Reprezentare:

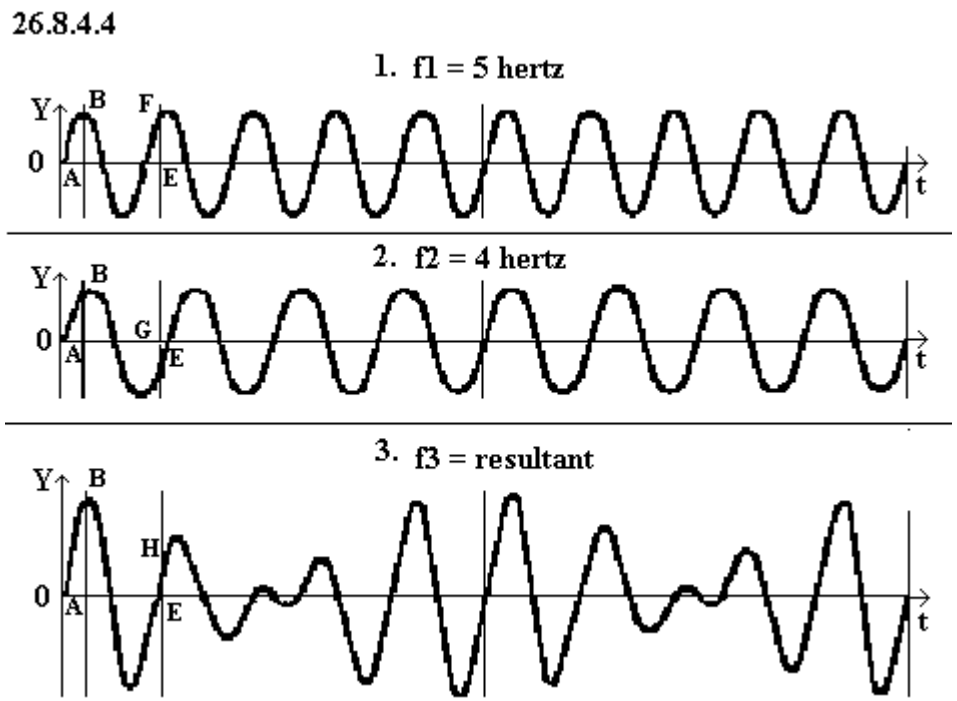
- Axa X: timp
- Axa Y: presiune (0 – presiunea aerului în repaus)

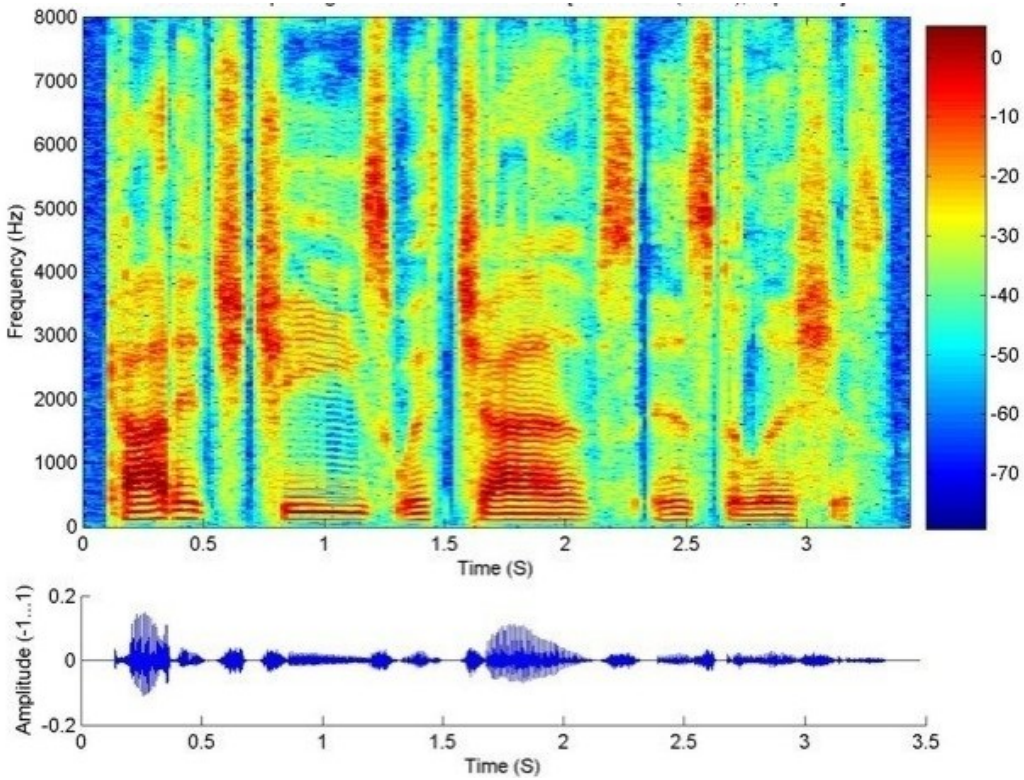
Amplitudine: măsoară dimensiunea vibrației / volumul sunetului

Frecvență: măsoară viteza vibrației / tonul sunetului



Compunere / descompunere





2. Numerizarea sunetului

Presupune stocarea și prelucrarea sunetului în format digital

Etape:

- Convertirea sunetului în semnal electric
- Eșantionarea și cuantificarea semnalului
- Stocarea informației numerice pe un suport de memorie externă conform unui format

Avantaje

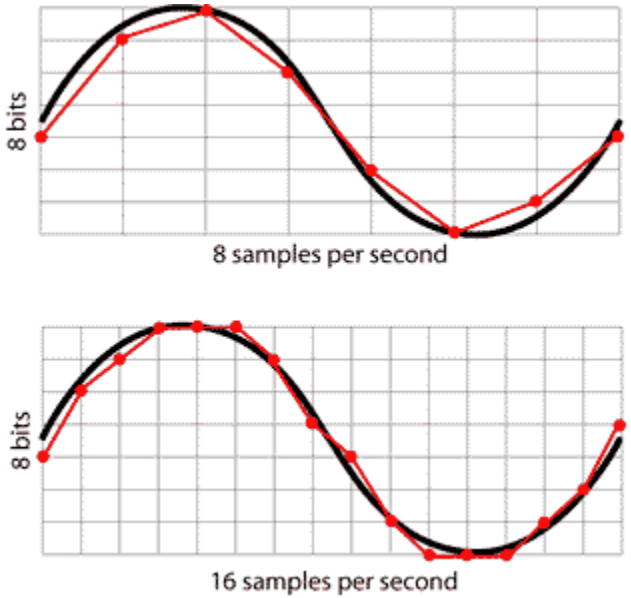
- Stocare mai ușoară
- Permite analiza și procesarea numerică a sunetului
- Nu se degradează în timp sau la copieri repetate

Eșantionare

Prin eşantionare se înţelege procesul de segmentare, cu o perioadicitate fixă, a semnalului audio analog.

Frecvența de eșantionare – rezoluția orizontală

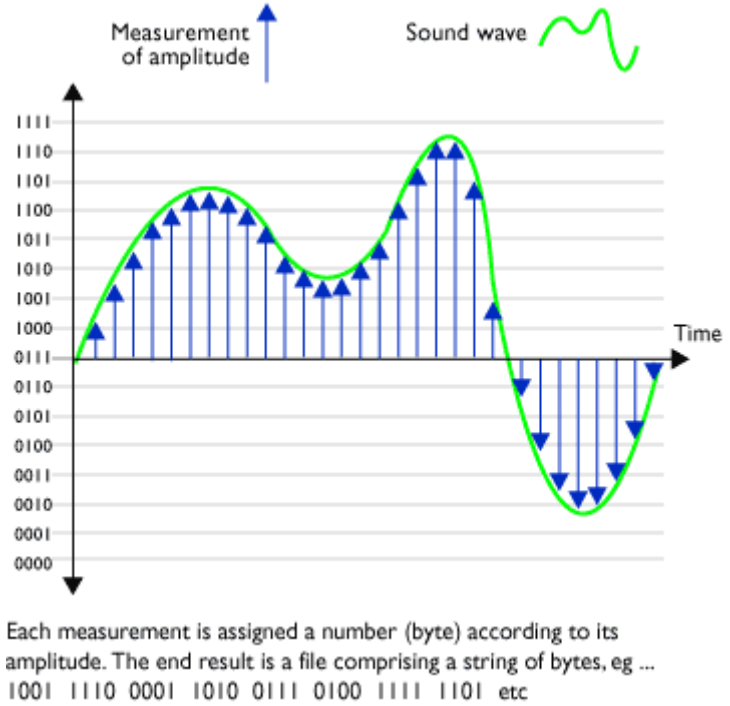
- Se determină pe baza teoremei lui Nyquist (minim dublul frecvenței maxime a sunetului)
- Rate de eșantionare uzuale:
  - 8 kHz – semnal telefonic
  - 11 kHz – radio AM
  - 22 kHz – radio FM
  - 44 kHz – audio CD



Cuantificare

Cuantificarea presupune asocierea unei valori numerice corespunzătoare amplitudinii semnalului pentru fiecare interval de timp.

Calitatea este influențată de numărul de biți alocați pentru fiecare eșantion (uzual 8 sau 16 biți pentru stocare și 16 – 32 pentru procesare)



Redarea sunetului digital:

- Se reconstruiește sinusoida originală prin interpolarea valorilor numerice stocate
- Prin intermediul unui convertor digital - analog

3. Formate audio

WAVE – formatul standard de fişier audio pentru Microsoft şi IBM; conţine sunet în reprezentare PCM necomprimat;

AIFF (Audio Interchange File Format) – formatul standard pentru audio digital utilizat pe platformele Apple (variante: necomprimat / comprimat);

MPEG (Moving Picture Experts Group) Audio - format standard pentru sunetul digital comprimat; parte a standardului MPEG de codificare a semnalului audio-video; cea mai cunoscută variantă a lui este MP3

OGG/VORBIS – algoritm de compresie perceptuală, similar MP3, dar cu implementare open source; folosește formatul de fișier OGG

4. Compresia sunetului

Cel mai utilizat algoritm de compresie: **MPEG-1 sau 2 Audio Layer III (MP3)**

Folosește codificare perceptuală

- Elimină din rezultat sunetele care nu pot fi percepute de către urechea umană

Sunetele imperceptibile sunt eliminate pe baza unui model psihoacustic care exploatează fenomenele de:

- Mascare a frecvențelor
- Mascare temporală

Compresia sunetului - mascarea

Mascarea frecvențelor

- Sunt eliminate sunetele cu frecvența mai mare de 16-18 KHz
- Sunt eliminate sunetele de intensitate scăzută, care apar concomitent cu sunete de intensitate înaltă, dacă sunt în benzi de frecvență alăturate (cele cu intensitate scăzută sunt mascate de cele cu intensitate înaltă)

Mascarea temporală

- Se elimină sunetele de intensitate mică care urmează după sunete de intensitate puternică în cadrul unui interval de timp
- Sunetele de intensitate mică nu pot fi percepute după sunete de intensitate puternică datorită inerției timpanului

Compresia MP3 - Etape

1. Utilizarea de filtre pentru separarea sunetului în 32 sub-benzi de frecvență

2. Transformări
   - FFT: se aplică modelul psihoacustic pentru determinarea factorului de scalare
   - DCT: proces de cuantizare cu factorul de scalare determinat anterior

3. Codificare Huffman pentru valorile cuantizate

4. Compunerea fluxului final de biți

5. Sunetul în context Web

La nivel de HTML – tag <audio>

Exemplu:

```
<audio controls="controls">
    <source src="test.wav" type="audio/wav">
    <source src="test.mp3" type="audio/mpeg">
</audio>
```

Formate suportate:

- mp3 – type = audio/mpeg
- wav – type = audio/wav
- ogg – type = audio/ogg

Audio - atribute

Elementul <audio>:

autoplay (bool) – redarea automată a sunetului

controls (string) – controalele de redare sunt afișate dacă atributul este prezent

loop (bool) – permite redarea continuă a sunetului

src (string)– permite specificarea sursei fără utilizarea de tag-uri de tip *source*

Elementul <source>

src (string) – adresa (URL) fișierului audio

type (string) – tipul MIME pentru fișierul audio

Audio – obiectul *HTMLMediaElement*

Proprietăți:

currentSrc – URL-ul absolut al fișierului redat

currentTime – poziția (în secunde) în cadrul fișierului (poate fi modificată)

duration – durata totală a fișierului audio (în secunde)

ended – boolean setat pe *true* la terminarea redării

error – ultima eroare (obiect *MediaError*) sau *null* dacă nu a apărut nici o eroare

paused – boolean setat pe *false* la oprirea redării

readyState – indică starea curentă a elementului

volume – permite citirea / modificarea volumului

Audio – obiectul *HTMLMediaElement*

Metode:

canPlayType(type) – permite aplicației să determine dacă browser-ul curent suportă un anumit tip de fișier audio

load() – pornește procesul de descărcare a fișierului audio de pe server; este obligatoriu să fie apelat înainte de începerea redării folosind metoda *play()*

pause() – oprește redarea (cu păstrarea poziției curente)

play() – pornește redarea de la poziția curentă

Audio – obiectul *HTMLMediaElement*

Evenimente:

canplay – a fost încărcată o parte din fișier și poate fi pornită redarea

ended – redarea s-a terminat

pause – redarea a fost oprită

play – redarea a început

volumechange – modificare de volum

waiting – operația curentă este suspendată pentru a încărca date de pe server

Web Audio API

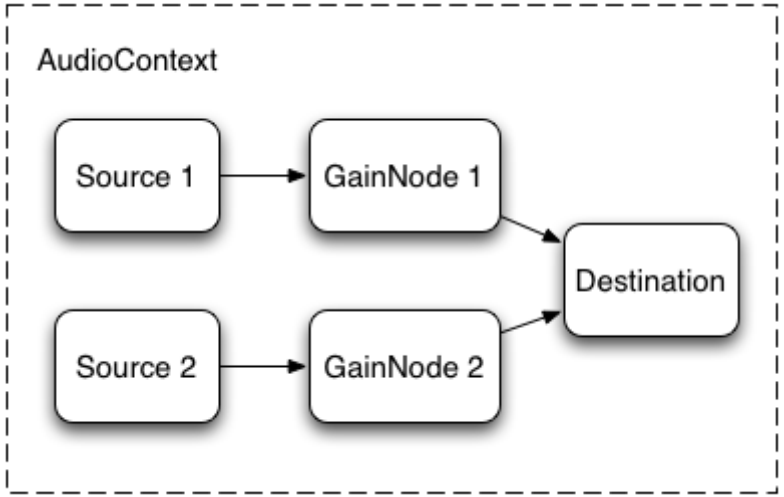
Permite generarea și procesarea de sunet în cadrul unui browser web.

Operațiile:

- se desfășoară în interiorul unui **context audio**
- sunt specificate prin intermediul unui **graf de rutare**

Tipuri de noduri:

- Sursă: preiau datele audio dintr-un tag audio sau generează sunete
- Procesare: prelucrează datele primite la intrare
- Destinație: nodul final al graficului



Web Audio API

AudioContext – reprezintă un graf de procesare audio

- Proprietăți: .currentTime (poziția curentă în secunde), .destination (referința la nodul destinație)
- Metode:
  - **createMediaElementSource(audioElement)**: construiește un nod sursă pe baza unui element <audio>
  - **createOscillator()**: construiește un nod sursă pentru generarea de sunete
  - **createGain()**: construiește un nod de procesare pentru ajustarea volumului
  - **createAnalyser()**: permite analiza sunetului (descompunre Fourier)
  - **createScriptProcessor(bufferSize)**: construiește un nod pentru procesare JavaScript
  - **.close()**:

AudioNode – reprezintă un nod din cadrul grafului (din care sunt moștenite celelalte noduri)

- .connect(node): conectează ieșirea nodului curent la intrarea nodului primit ca parametru
- Diverse metode și proprietăți în funcție de tipul nodului

IV. Video

1. Noțiuni generale
2. Compresia video
3. Video în context web

1. Noțiuni generale

Video digital – cuprinde totalitatea tehnicilor de captură, procesare și stocare a imaginilor în mișcare (precum și a sunetului asociat) prin intermediul unui dispozitiv de calcul.

Avantaje video digital:

- Poate fi procesat prin intermediul calculatorului
- Păstrare în timp și rezistență la copieri repetate
- Poate fi transmis la distanță

Caracteristici video

Rezoluția

Spațiul de culoare și numărul de biți per pixel

Numărul de cadre pe secundă

Modul de afișare (întrețesut sau progresiv)

Calitatea compresiei

Formate video

Container – specifică structura de stocare a componentelor video (imagine + audio) și a datelor asociate (metadate, subtitrări, …)

- Advanced Systems Format – **ASF**: container dezvoltat de Microsoft care poate conține fluxuri codate cu orice codec (Extensii: .asf, .wma, .wmv)
- Audio Video Interleave – **AVI**: container mai vechi dezvoltat de Microsoft pe baza Resource Interchange File Format – RIFF (stochează datele în secțiuni identificate prin markere FourCC)
- **MP4** – MPEG-4 Part 14: dezvoltat de către Motion Pictures Expert Group și utilizat inițial de către QuickTime (video H.264, audio AAC)
- **AVCHD** – format utilizat în special de către camerele video (video H.264 AVC și sunet AC3 sau PCM)
- **Matroska / OGG**: formate deschise; pot conține mai multe fluxuri audio / video

Formate video

Codec – specifică modalitatea de compresie / decompresie pentru un flux video / audio în cadrul unui container

- **H.264 / MPEG-4 AVC** – cel mai popular (utilizat pentru Web, BluRay, camere video)
- **H.262 / MPEG-2** – formatul standard pentru DVD
- **Windows Media Video** – format dezvoltat de către Microsoft
- **MJPEG (Motion JPEG)** – format mai vechi bazat pe compresia JPEG

2. Compresia video

Se bazează pe reducerea redundanței din cadrul fluxului video

Redundanța spațială (intra-cadru)

- tipul de redundanță identificat și eliminat de algoritmii de compresie a imaginilor

Redundanța temporală (inter-cadru)

- redundanță identificată între două cadre consecutive (de exemplu, prin compararea a două cadre se observă că majoritatea pixelilor își păstrează valoarea)

Compresia MPEG

Algoritm de compresie video

- Hibrid
  - Transformata Cosinus Discretă – similar JPEG pentru reducerea redundanței spațiale
  - Codaj Huffman – pentru comprimarea coeficienților TCD
  - Codificarea mișcării – pentru reducerea redundanței temporale
  - Codaj RLE
- Asimetric
  - Timpul de codare este mult mai mare decât cel de decodare

Etapele de compresie

Împărțirea imaginii în blocuri

- 16x16 luminanță
- 8x8 crominanță (culoare)

Compresie pe baza DCT pentru reducere spațială

Aplicarea tehnicilor de compensare a mișcării pentru reducere temporală

Faza finală de codare pe două dimensiuni folosind Run Length Encoding

Tipuri de cadre

<I> *Intra-picture/frame/image*

- Cadrele cheie
- Necesare pentru căutare și poziționare
- Compresie moderată

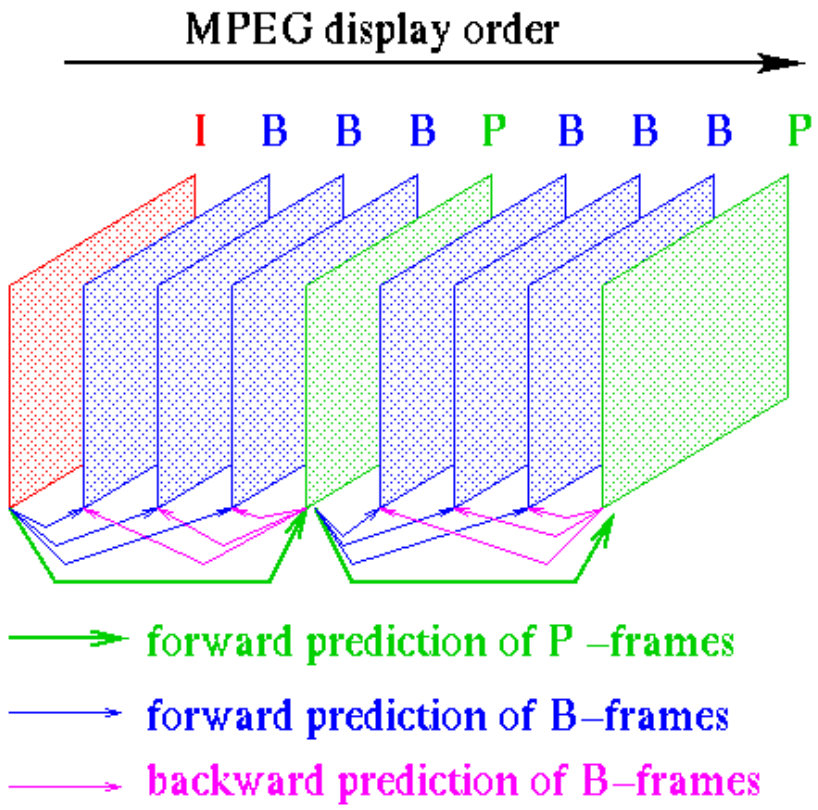
<P> *Predicted pictures*

- Codate cu referință la un cadru anterior
- Folosite ca referință pentru cadre ulterioare

<B> *Bi-directional prediction (interpolated pictures)*

- Necesită cadre anterioare și viitoare pentru refacere
- Compresie mare

Tipuri de cadre



4. Video în context Web

La nivel de HTML – tag <video>

Exemplu:

```
<video controls="controls">
    <source src="test.webm" type="video/webm">
    <source src="test.mp4" type="video/mp4">
</video>
```

Formate suportate:

- webm – type = video/webm
- mp4 – type = video/mp4
- .ogg – type = video/ogg

Atribute / manipulare din JavaScript

Similar cu elementul audio

- Atribute: *autoplay, controls, src, volume, ...*

Reprezentat la nivel DOM prin obiecte de tip HTMLMediaElement

- Proprietăți: currentSrc, currentTime, duration, ended, error, paused, readyState, volume
- Metode: canPlayType, load, pause, play
- Evenimente: canplay, ended, pause, play, volumechange, waiting

Exemple: Încărcare dinamică video

```
var v = $("<video></video>")
            .attr({
               "controls": "hidden",
               "autoplay": "autoplay",
               "src": "media/movie.mp4"
            })
            .load()
            .appendTo($("body"));

// modificare sursă
v[0].src = "media/test.mp4";
v[0].load();
v[0].play();
```

Exemple: Procesare cadru - 1

```
var video = $("#video")[0];

var canvas = $("#canvasProcesare")[0];

var context = canvas.getContext("2d");

function procesareCadru() {
        //…
        requestAnimationFrame(procesareCadru);
 };
 requestAnimationFrame(procesareCadru);
```

Exemple: Procesare cadru - 2

```
var W = canvas.width = video.clientWidth;
var H = canvas.height = video.clientHeight;

context.drawImage(video, 0, 0, W, H);
var imageData = context.getImageData(0, 0, W, H);

for (var y = 0; y < H; y++) {
   for (var x = 0; x < W; x++) {
      var i = (y * W * 4) + x * 4;
            // modificare valori imageData.data[i+...]
      }
   }
}
context.putImageData(imageData, 0, 0);
// alte operații de desenare pe canvas
```

Exemple: Playlist simplu

```
$(function () {
        var lista = ["movie.mp4", "v2.mp4"];
        var index = 0;

        var video = $("#myVideo");

        video.on("ended", function () {
            index = index + 1;
            if (index >= lista.length) {
                index = 0;
            }
            video[0].src = lista[index];
            video[0].load();
            video[0].play();
        });
    });
```