Department of Economic Informatics and Cybernetics
Bucharest University of Economic Studies

Multimedia

Liviu-Adrian Cotfas, PhD.

liviu.cotfas@ase.ro

Few words about me…

https://ro.linkedin.com/in/cotfasliviu

Aspecte administrative

- slide-urile marcate în laterală cu "Opțional" conțin informații adiționale, care nu vor fi solicitate la evaluare (lucrare, proiect și colocviu);
- în acest context, unele dintre slide-urile marcate în laterală cu "Opțional" pot fi redactate în limba engleză;
- slide-urile în limba engleză, nu vor fi solicitate la evaluare (lucrare, proiect și colocviu);

Recommended Reading / Watching

- Prezentare curs, Exemple seminar:
  - https://github.com/liviucotfas/ase-multimedia
  - http://online.ase.ro

Opțional

Further Reading / Watching

- https://web.dev

- Cursuri **Microsoft Virtual Academy** - mva.microsoft.com
  - Free

- Cursuri **PluralSight** - www.pluralsight.com
  - Free trial
  - Free access (limited period) through Microsoft DreamSpark

Definiție, Caracteristici, Concepte

Definiție

- **Multimedia** este din punct de vedere informatic :
  - orice combinație de **text**, **imagine**, **video**, **sunet** și **animație**
  - accesibilă utilizatorului prin intermediul sistemului de calcul.

- **Dezvoltarea multimedia** a devenit posibilă ca urmare a Revoluției Digitală, prin intermediul :
  - conversiei analog-digital;
  - compresiei datelor.

Factori declanșatori

- **Revoluția digitală** reprezintă schimbarea de la tehnologia mecanică și electronică analogică la electronica digitală care a început la sfârșitul anilor 1950 până la sfârșitul anilor 1970, odată cu adoptarea și proliferarea computerelor digitale și păstrarea înregistrărilor în format digital.

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

- Există două categorii majore de algoritmi de compresie :
  - Compresie lossless (fără pierderi)
  - Compresie lossy (cu pierderi)
- Algoritmii de compresie utilizați în multimedia sunt de obicei **asimetrici** – timpul de compresie este mai mare decât timpul de decompresie.

Compresia lossless

- utilizează o codificare mai eficientă pentru a reduce dimensiunea fișierului, păstrând în același timp toate datele originale. Când fișierul este decomprimat, acesta va fi identic cu originalul.
- Run Length Encoding - RLE
  - una dintre strategiile mai simple pentru a realiza compresia lossless
  - poate fi folosit pentru a comprima fișiere de imagine bitmap(ex: format *pcx). Imaginile bitmap pot deveni cu ușurință foarte mari, deoarece fiecare pixel este reprezentat cu o serie de biți care oferă informații despre culoarea sa. RLE generează un cod pentru a „semnaliza" începutul unei linii de pixeli de aceeași culoare. Informațiile despre culoare sunt apoi înregistrate o singură dată pentru fiecare pixel. De fapt, RLE spune computerului să repete o culoare pentru un anumit număr de pixeli adiacenți, mai degrabă decât să repete aceleași informații pentru fiecare pixel de mai multe ori. Fișierul comprimat RLE va fi mai mic, dar va păstra toate datele originale ale imaginii – este "lossless".

Run Length Encoding - RLE

Compresia Lossy

- numărul de biți din fișierul original este redus și unele date se pierd. Compresia Lossy nu este o opțiune pentru fișierele constând din text și numere, așa-numitele informații *alphanumerice*. În cazul acestora pierderea unei singure litere sau număr ar putea modifica cu ușurință semnificația datelor

- este adesea posibil să se mențină imagini sau sunete de înaltă calitate cu mai puține date decât era prezent inițial (util în special pentru multimedia)

- exploatează limitele percepției umane

Compresia Lossy

- Example:
  - JPEG – imagini;
  - MPEG – sunet și video.
- compresie MP3:
  - analizează fișierul audio și elimină date care nu sunt critice pentru o redare de înaltă calitate;
  - elimină frecvențele care nu pot fi percepute de auzul uman. De asemenea, poate evalua două sunete care rulează în același timp și poate elimina sunetul mai puțin perceptibil. Aceste tipuri de date pot fi eliminate fără impact semnificativ asupra calității;
  - poate reduce cu un factor de 10 cantitatea de date necesară pentru a reprezenta înregistrările audio digitale, la o calitate similară cu cea a sunetului original necomprimat pentru majoritatea ascultătorilor.

Aplicații Multimedia

- Caracteristici:
  - 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.

Abordări pentru construirea de aplicații multimedia

- Multimedia authoring – high-level
  - 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 – low-level
  - accentul se pune pe procesarea directă a resurselor prin intermediul unui limbaj de programare precum C, C#, JavaScript sau Java și a unor biblioteci specializate.

Classification of multimedia applications

- Mai multe criterii de clasificare bazate pe :
  - domeniu: economie, educație, publicitate, medicină, industrie, divertisment, sisteme de navigație și informații, comunicații;
  - interactivitate: interactiv, static;
  - localizarea componentelor: local, la distanță (video-streaming, distant learning).

Precondiții hardware / software

Precondiții hardware

- Echipamente hardware pentru achiziție / procesare:
  - Imagine;
  - Sunet;
  - Video.

Dispozitive periferice - Imagine

- Scanner
  - transformă informația luminoasă în informație electrică, iar ulterior aceasta este convertita și salvată sub formă digitală.
  - tipuri: flatbed / rotativ
  - operation: o lumină trece peste text sau imagine, iar lumina se reflectă înapoi pe un senzor CCD (charge-coupled device). Un senzor CCD este un dispozitiv electronic care captează semnaulul analogic. Semnalul analogic este apoi convertit într-un semnal digital de către un alt dispozitiv numit an ADC (analog-to-digital converter) și transferat prin intermediul unei interfețe (în general USB) către RAM.
  - Recunoașterea optică a caracterelor (OCR) este procesul de conversie a textului tipărit într-un fișier digital care poate fi editat într-un procesor de text.

Dispozitive periferice - Imagine

- scanner

Dispozitive periferice - Imagine

- Camere foto
  - Când declanșatorul camerei este deschis pentru a captura o imagine, lumina trece prin obiectivul camerei. Imaginea este focalizată pe un CCD, care generează un semnal analogic.
  - Semnalul analogic este convertit în formă digitală de un ADC și apoi trimis la un procesor de semnal digital (DSP) chip care ajustează calitatea imaginii și o stochează în memoria încorporată a camerei sau pe un card de memorie.

Dispozitive periferice - Sunet

- Placă de sunet
  - convertește semnalul analogic de la microfon la o reprezentare digitală;
  - convertește reprezentarea digitală într-una analogică care poate fi redată de difuzoare.

Dispozitive periferice - Video

- Placă video
- Cameră video digitală

Precondiții software

- **Drivere** - program de computer care operează sau controlează un anumit tip de dispozitiv care este atașat la un computer. Un driver oferă o interfață software pentru dispozitivele hardware, permițând sistemelor de operare și alte programe de computer să acceseze funcțiile hardware fără a fi nevoie să știe detalii precise despre hardware-ul utilizat.

- Extensii ale sistemului de operare:
  - blioteci specializate pentru controlul resurselor multimedia;
  - instrumente de bază pentru redarea conținutului (ex: Groove Music).

- Software specializat

Software specializat

- Există două tipuri majore de software pentru dezvoltare multimedia:
  - **Media-specific** applications sunt folosite pentru a crea și edita elementele media individuale (text, grafică, sunet, video, animație) care alcătuiesc un produs multimedia.
  - **Authoring** applications conțin instrumente software pentru integrarea componentelor media și furnizarea unei interfețe cu utilizatorul.

Grafică

- Achiziția și prelucrarea de imagini 2D sau 3D.
- Categorii:
  - prelucrare imagini raster;
  - prelucrare imagini vectoriale;
  - 2D / 3D.

Aplicații prelucrare imagini raster

- conțin seturi de instrumente pentru a crea obiecte grafice, precum și instrumente de editare pentru fotografii digitale sau imagini scanate

- oferă o gamă largă de caracteristici, cum ar fi filtre (blur, emboss, pixelate), setări de reglare a imaginii (redimensionare, luminozitate, rotație) și efecte (umbră, gradient overlay).

- asigurarea unui control asupra elementelor individuale ale imaginii este posibilă folosind straturi (en: layers) și setări de mască. Instrumentele de tip text sunt utilizate pentru a genera text grafic cu modele, forme și efecte 3D.

Aplicații prelucrare imagini raster

- Example: **Photoshop** (Adobe), Gimp (open source)

Aplicații prelucrare imagini vectoriale

- conține un set distinct de instrumente pentru crearea de forme de bază, cum ar fi ovale, dreptunghiuri, curbe Bezier și poligoane generate utilizând formule matematice.

- formele pot fi grupate, umplute și redimensionate pentru a produce imagini complexe.

- astfel de aplicații pot fi folosite pentru a crea obiecte grafice care pot fi redimensionate cu ușurință pentru proiecte multimedia diverse.

- Example: Illustrator (Adobe), Draw (Corel), Inkscape (open source)

Aplicații prelucrare imagini vectoriale

Aplicații prelucrare imagini 3D

- folosit pentru a modela obiecte 3D, a defini suprafețe, a compune scene și a exporta imaginea rezultată;

- În etapa de modelare (en: ***modeling***), se creează forma obiectului; în etapa ulterioară, de definire a suprafeței (en: ***surface definition***), se aplică culoarea și textura; în etapa de construire a scenei (en: ***scene composition***), obiectele sunt aranjate în cadru, iluminate, se adaugă fundalul și efectele speciale. Ultima etapă, este reandarea (en: ***rendering***). Randarea creează o imagine pornind de la o scenă 3D. Rendering is both processor intensive and time consuming because the software must calculate how the image should appear based on the object's position, surface materials, lighting, and specific render options.

- Example: Blender (open source)

Aplicații prelucrare imagini 3D

Sunet

- Există două tipuri majore de aplicații de sunet pentru dezvoltarea multimedia:
  - *sunet eșantionat (en: sampled);*
  - *sunet sintetizat (en: synthesized).*
- Example: Audition (Adobe)

Sunet eșantionat

- sunetele eșantionate sunt reprezentări digitale ale surselor de sunet analogice capturate de la microfoane sau alte dispozitive.

- sunetele eșantionate pot fi editate într-o mare varietate de moduri, cum ar fi tăierea pentru a șterge porțiunile goale, îmbinarea pentru a combina segmente de sunet, setarea fade-in și fade-out, reglarea volumului și adăugarea de efecte speciale, cum ar fi ecouri sau inversări de sunet

Sunet sintetizat

- aplicațiile de sunet sintetizat folosesc comenzi digitale pentru a genera sunete. Aceste comenzi pot fi capturate de la un instrument MIDI, cum ar fi o tastatură electronică sau create cu un program specializat.

Sunet

Adobe Audition
Waveform
Multitrack
Workspace: Default
Search Help
Files
Name
Status
Duration
Sample Rate
gd90-09-14d2.flac
56:59.768
44100 Hz
Editor: gd90-09-14d2.flac
Scarlet Begonias
Fire on the Mtn
Truckin'
Terrapin Station
Jam
Media Browser
Markers
Properties
Start
End
Type
Scarlet Begonias 0:00.000 10:58.546 10:58.546 CD Track
Fire on the Mtn 10:58.546 26:02.200 15:03.653 CD Track
Truckin' 26:02.200 50:57.160 24:54.960 CD Track
Terrapin Station 36:23.240 50:57.160 14:33.920 CD Track
Jam 50:57.160 56:59.773 6:02.613 CD Track
Drums 56:19.986 56:59.773 0:39.786 CD Track
Cue
Subclip
CD Track
Cart Timer
History
Open
0 Undo
0:00.000
Save completed in 12.65 seconds
44100 Hz • 32-bit (float) • Stereo
1.12 GB
56:59.768
102.97 GB free

Video

- un mediu pentru a combina videoclipuri sursă, pentru a sincroniza clipurile cu o piesă sonoră, pentru a adăuga efecte speciale și pentru a salva rezultatul ca un videoclip digital.

- un proiect video începe prin încărcarea resurselor utilizate într-o fereastră de proiect. Resursele pot consta în imagini, animații, sunete, fișiere video.

- aplicațiile video oferă instrumente pentru a muta și insera clipuri pe o axă a timpului, pentru a tăia clipul și a defini tranziții între piese. Aplicațiile de editare video definesc, de asemenea, dimensiunea redării și rata de afișare a cadrelor. Când proiectul video este finalizat, aplicația oferă setări pentru salvarea acestuia utilizând diferite scheme de compresie..

Video

- Example: Premiere (Adobe)

Animație

- Folosit pentru a crea și edita secvențe animate. **Animația** este tehnica utilizării unei serii de imagini statice afișate rapid pentru a crea iluzia de mișcare.

- Fiecare cadru reprezintă o singură instanță a secvenței animate. Instrumentele tipice de animație controlează calea unui obiect, forma obiectului și modificările de culoare pe secvența de cadre.

- Obiectele sunt plasate pe o **axă a timpului** în care efectele pot fi aplicate pentru a se estompa, transforma, roti, răsturna sau schimba ritmul. Mai multe obiecte pot fi stratificate pentru a interacționa între ele pentru a crea animații mai complexe.

Animație

- Example: Director (Adobe), Flash (Adobe), Animate (Adobe)

Software pentru dezvoltare multimedia

- constă din programe special concepute pentru a facilita crearea de produse multimedia.
- acestea sunt utilizate pentru a combina elemente media, a sincroniza conținutul, a proiecta interfața utilizatorului și a oferi interactivitate utilizatorului.

Software pentru dezvoltare multimedia

- Categorii în funcție de abordarea utilizată pentru a organiza elementele multimedia:
  - card-based;
  - timeline;
  - diagramă de flux.

Card-based

- elementele sunt aranjate la fel cum ar putea fi pe fișele de index sau pe paginile unei cărți.
- astfel de aplicații sunt ușor de utilizat, în special pentru prelegeri și tutoriale care nu necesită o sincronizare precisă a elementelor media individuale
- Example: **PowerPoint**, ToolBook

Timeline

- Utilizarea unui *timeline* compus din cadre similar cu un film;
- astfel de aplicații oferă controlul precis necesar pentru animații avansate;
- Example: Director (Adobe), Flash (Adobe), Animate(Adobe).

Adobe Flash

File Edit View Insert Modify Text Commands Control Debug Window Help
ESSENTIALS
badge*
00.exe*
Page 1
100%
PROPERTIES
LIBRARY
00.exe
688 items
Name
Linkage
Buttons
Button 476
Button 477
Fonts
Font 191
Glyph 0
Glyph 1
Glyph 2
Glyph 3
Glyph 4
Glyph 5
Glyph 6
Glyph 7
Glyph 8
Font 2
Images
Image 193
Image 195
Image 460
Morphs
Shapes
Sounds
Sprites
Texts
TIMELINE
MOTION EDITOR
1280 1285 1290 1295 1300 1305 1310 1315 1320 1325 1330 1335 134
Layer

Evoluția Adobe Flash

Diagramă de flux

- utilizează diagrame de flux pentru a dezvolta rapid o gamă largă de produse multimedia, inclusiv tutoriale avansate, demonstrații de produse și simulări. Pictogramele pot reprezenta atât conținut (imagini, text, animații, video), cât și o gamă largă de interacțiuni (redare, oprire, accesare, calculare etc.).

- example: Authorware

Multimedia în context WEB

Multimedia în context WEB

- HTML5 oferă un suport mult îmbunătățit pentru multimedia web, prin includerea elementelor precum audio, video, canvas

- elemente multimedia suportate:
  - text
  - imagini
  - animație
  - grafică raster - elementul <canvas>
  - grafică vectorială – elemental <svg>
  - 3D graphics - WebGL
  - audio – elementul <audio>
  - video – elementul <video>

HTML5 Games

http://www.cuttherope.net/basic_standalone/game.html

HTML5 Videos

Limbajul JavaScript

Caracteristici

- Dinamic
- Bazat pe obiecte

Utilizare

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

Modalități de includere în HTML

Prin intermediul tag-ului <script>:

1. Fișier extern

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

2. În cadrul paginii

```
<script>
// cod JavaScript …..
let test = 10;
</script>
```

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 }

Further reading: https://www.w3schools.com/js/js_datatypes.asp

Tipuri de date

Obiecte speciale

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

Variabile și expresii

Declarare

- Prin atribuire valoare (**nerecomandat**): a = 8; a = *false*;
- Folosind **var (nerecomandat)**, **let** sau **const:**

```
var a = 3; var b;
let b = "test"; let b;
const b = "test"
```

Exemplu:

```
const c = 7; // c = 3; //error
const v = [1, 2, 3]; v[0] = 5; //works fine
```

Variabile și expresii

Scope:

- Global sau local
- Function vs block based

Evaluare expresii

- operatori similari cu C# (în plus ===, !==)

Exemple:

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

Vectori – Obiectul *Array*

Inițializare: *var v = []*; sau *var v = [1, "lon"]*

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

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

Metode:

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

Funcții

Declarare:

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

var suma = *function*(a, b) { *return* a + b; } // fără hoisting

Parametri:

Transmiși prin valoare

Accesibili prin *arguments*

Apel - nume_funcție / expresie(parametri):

var rezultat = suma(7, 3);

var test = *function(*val) { return val + 1;}(5);

Obiecte

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

Declarare obiecte

Literali: var ob = { nume: "Ana", varsta:7 }

new: let 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;

Obiecte

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* Persoana(nume) { this.nume = nume; this.varsta = 1; }
- Apel: let ob2 = new Persoana("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();

Document Object Model (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* / *nextElementSibling*)
- Conține metode pentru manipularea nodurilor copil: *appendChild*(nod), *removeChild*(nod), ...

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

Rădăcina: *document.documentElement*

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

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");

Accesare atribute:

elem.numeAtribut – accesare valoare atribut individual

elem.attributes – colecția de atribute

Accesare atribute CSS:

elem.style.numeAtributCSS

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</a>*

- Prin proprietăți nod:

  element.proprietateEveniment = funcție;

  Exemplu:

  let elem = document.getElementById("test");

  elem.**onclick** = function() {console.log("un mesaj");}

Tratare evenimente

- Prin metoda addEventListener:

element.*addEventListener*(tipEveniment, funcție);

Exemplu:

```
document.getElementById("test")
        .addEventListener(
                "click",
                function() {console.log("un mesaj");}
        );
```

Eliminare event handler:

```
element.removeEventListener(tipEveniment, funcție);
```

Tratare evenimente

Accesare element sursă – *this*

Parametri eveniment – obiect *event*

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

Evenimente

- General: *load*
- Tastatură: *keydown*, *keypress*, *keyup*
- Mouse: *mouseenter*, *mouseleave*, *mousemove*, *mousedown*, *mouseup*, *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

JavaScript

- http://jstherightway.org/

Culori

Lumina

Increasing energy
Increasing wavelength
0.0001 nm
0.01 nm
10 nm
1000 nm
0.01 cm
1 cm
1 m
100 m
Gamma rays
X-rays
Ultra-violet
Infrared
Radio waves
Radar
TV
FM
AM
Visible light
400 nm
500 nm
600 nm
700 nm

Lumina

- ne referim la lumina soarelui naturală ca lumină albă, deoarece ochiului uman o percepe ca fiind incoloră.

- este posibilă separarea luminii albe cu ajutorul unei prisme. La trecerea luminii printr-o prismă, aceasta este descompusă în lungimile de undă ale culorii sale componente.

Lumina

Culorile primare și secundare ale luminii albe devin vizibile atunci când sunt refractate de o prismă de sticlă.

Lumina

O suprafață verde absoarbe toate culorile, cu excepția verdelui, pe care îl reflectă.

Lumina

[1] http://www.mstworkbooks.co.za/natural-sciences/gr8/gr8-ec-04.html.

Lumina

- Deși putem obține vopsea neagră ca pigment, negrul nu este o culoare a luminii. Negrul este rezultatul absorbției complete a luminii.

Modele de culoare

Model aditiv(RGB)

Model subtractiv (CMY / CMYK)

Model aditiv

- În **modelele aditive** culorile sunt create prin amestecarea mai multor culori primare.

- Roșu, verde și albastru sunt culorile primare cel mai des utilizate în modelul aditiv.

- Modelul aditiv pornește de la **negru** și ajunge la **alb**; pe măsură ce se adaugă mai multă culoare, rezultatul este mai deschis și tinde spre **alb**.

- Combinarea a **două** dintre cele **trei** culori primare standard utilizate în modelul aditiv în proporții egale produce una dintre culorile secundare din modelul aditiv - **cian**, **magenta** sau **galben**

Modelul RGB

- utilizează culorile de bază **roșu**, **verde** și **albastru** (RGB).
- reglând intensitatea fiecărei culori se pot obține toate culorile din spectrul luminii vizibile. Se va obține culoarea albă prin combinarea culorilor de bază în mod egal.
- Dacă ar fi să ne uităm la un monitor LCD (ecran cu cristale lichide) la microscop, am vedea că fiecare pixel afișează în realitate doar cele trei culori primare ale luminii.

Modelul RGB

- Prin combinarea culorilor roșu și verde se obține culoarea galben. Dacă umplem o formă geometrică cu galben în Photoshop, pixelii aferenți zonei respective vor avea active doar componentele de roșu și verde (componenta de albastru nu va fi activă).

- Punctele individuale de culoare sunt în cazul ecranelor LCD suficient de mici, încât creierul uman combină culorile și le percepe ca galben.

Modelul RGB

Adobe Photoshop – Modelul RGB

Modelul RGB

Combinațiile de culori posibile pentru un pixel, utilizând 8 biți pentru fiecare culoare (afișaj pe 24 de biți).

Modelul RGB

- 256 de valori posibile pentru fiecare canal de culoare.

- 256 de valori posibile pentru roșu × 256 pentru verde × 256 pentru albastru - **16.8 milioane** de combinații posibile / culori.

HSL - Hue, Saturation and Lightness Color Model

- Reprezentare sub formă de coordonate cilindrice

- Bazat pe aceleași culori de bază:
  - Roșu - 0^0
  - Verde - 120^0
  - Albastru - 240^0

- Cele două reprezentări rearanjează geometria RGB într-o încercare de a o face mai intuitivă și mai relevantă perceptual decât reprezentarea cartesiană (cub)

HSL - Hue, Saturation and Lightness Color Model

- Roșu - 0^0
- Verde - 120^0
- Albastru - 240^0

HSL - Hue, Saturation and Lightness Color Model

- HSL și HSV sunt utilizate în instrumentele de selectare a culorii, în analiza imaginilor și computer vision.

Instrument de selecție a culorii în Paint.NET:
http://www.getpaint.net/index.html

Modele substractive

- Combinarea culorilor în procesul de imprimare utilizează metoda **substractivă** [1].

- **Modelul substractiv** explică cum este posibil ca prin combinarea unui număr limitat de pigmenți sau coloranți naturali să se obțină o paletă largă de culori. Adăugarea fiecărei culori determină reducerea parțială sau completă (absorbția) unor lungimi de undă a luminii (și reflectarea celorlalte)

- În acest model se pornește de la culoarea albă și se ajunge la negru; pe măsură ce se adaugă culoare, rezultatul devine mai întunecat și tinde spre negru [2].

[1] https://en.wikipedia.org/wiki/Subtractive_color
[2] http://www.worqx.com/color/color_systems.htm

Modelul CMYK

- Modelul de culoare **CMYK** este utilizat în procesul de imprimare[1].
- CMYK se referă la cele patru cerneluri utilizate în majoritatea imprimărilor color: cyan, magenta, yellow și key (black) [1].
- Deși variază în funcție de tipografie, producător, imprimantă, cerneala este de obicei aplicată în ordinea abrevierii[1].

Modelul CMYK

https://commons.wikimedia.org/wiki/File:NIEdot367.jpg

Modelul CMYK

Cercul culorilor (en: Color Wheel)

- Cercul culorilor este o reprezentare vizuală a culorilor dispuse în funcție de **relațiile cromatice** dintre ele. Cercul culorilor este format prin poziționarea nuanțelor primare echidistant una față de cealaltă. Între culorile primare sunt apoi plasate culorile secundare și terțiare [1].

Color Picker in Paint.NET:
http://www.getpaint.net/index.html

Cercul culorilor (en: Color Wheel)

- **Culori primare:** culorile care nu pot fi create prin amestecarea altora[1].

- **Culori secundare:** acele culori realizate printr-un amestec de două culori primare [1].

- **Culori terțiare:** acele culori realizate printr-un amestec de nuanțe primare și secundare[1].

Cercul culorilor (en: Color Wheel)

- **Culori complementare:** acele culori situate diametral opus pe cercul culorilor [1].

- **Culori apropiate:** acele culori situate aproape una de cealaltă pe o cercul culorilor[1].

Active & Passive Colors

- The color wheel can be divided into ranges that are visually:
  - Active colors - will appear to advance when placed against passive hues.
  - Passive colors - appear to recede when positioned against active hues.

- Advancing hues are most often thought to have less visual weight than the receding hues.

- Most often warm, saturated, light value hues are "active" and visually advance.

Active & Passive Colors

- Cool, low saturated, dark value hues are "passive" and visually recede.
- Tints or hues with a low saturation appear lighter than shades or highly saturated colors.
- Some colors remain visually neutral or indifferent.

Complementary Colors

- When fully saturated complements are brought together, interesting effects are noticeable. This may be a desirable illusion, or a problem if creating visuals that are to be read.

Does this text have highlighted edges?

- Notice the illusion of highlighted edges and raised text. This may occur when opposing colors are brought together.

CSS – Specificarea colorilor

Format hexazecimal:

- *#rrggbb*
- *rr,gg,bb* sunt numere în baza 16

```
<h1 style="background-color:#ff6347;">...</h1>
```

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

```
<h1 style="background-color:rgb(255, 99, 71);">...</h1>
```

CSS – Specificarea colorilor

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

```
<h1 style="background-color:hsl(9, 100%, 64%);">...</h1>
```

Imaginea

Grafică pe computer 2-D

- Calculatoarele pot fi utilizate pentru a crea imagini 2D (lățime și înălțime) sau 3D (lățime, înălțime și adâncime).

- Există două tipuri principale de grafică computerizată 2D:

  - **bitmap -** potrivită pentru imagini cu detalii fine, cum ar fi picturi sau fotografii. Numită și **grafică raster.** Stochează informația despre culoarea fiecărui pixel.

  - **vectorială** - utilizat pentru desene grafice, de la desene și logo-uri simple la creații artistice sofisticate. Utilizează descrieri matematice.

Grafică pe computer 2-D

Grafică raster

Imiagini digitale

- O **imagine** este o reprezentare bidimensională sau tridimensională a unei persoane, obiect sau scenă din lumea naturală.

- O **imagine digitală** este o reprezentare numerică a unei imagini bidimensionale.

Grafică raster

- o **imagine** grafică **raster** este formată prin împărțirea zonei unei imagini într-o matrice dreptunghiulară de rânduri și coloane compuse din pixeli[1].

- un **pixel** este zonă pătrată (sau ocazional rectangulară) de culoare, reprezentând un singur punct din imagine [2].

0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	1	1	1	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0	0
0	1	1	1	0	0	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Grafică raster

- Toți pixelii dintr-o imagine raster au exact aceeași dimensiune. Fiecare specifică o singură culoare stocată utilizând 24 de biți.

- Numărul total de pixeli dintr-o imagine raster este fix. Pentru a mări o imagine raster este necesară adăugarea de pixeli suplimentari.

- Similar, anumiți pixeli trebuie eliminați le reducerea dimensiunilor unei imagini raster. Lățimea și înălțimea unei imagini raster sunt determinate de câte pixeli conține fiecare rând și coloană.

Mozaic

În cazul unui mozaic privit de la distanță, plăcile individuale care formează imaginea sunt abia perceptibile cu ochiul liber. Acestea devin însă vizibile dacă mozaicul este privit de aproape. Această tehnică, ce presupune utilizarea unor mici bucăți de material colorat a apărut cu aproximativ 3.000 î.Hr.

Mozaic din secolul al XIX-lea

Pixeli

Prin utilizarea facilității de zoom disponibilă în programele de grafică, pixelii individuali devin vizibili sub formă de pătrate.

Pixeli

Activitate practică:

1. Copiați imaginea de pe slide-ul anterior în editorul de imagini raster favorit (ex: MS Paint)
2. Măriți la maxim nivelul de zoom. Activați opțiunea gridlines, dacă este disponibilă.

Caracteristici

- Rezoluție
  - descrie calitatea unei imagini raster și se referă direct la dimensiunea și numărul de pixeli conținuți de imagine.
- Dimensiunea în pixeli
  - descrie dimensiunea unei imagini raster, exprimată ca numărul de pixeli de-a lungul axei x (lățime) și numărul de pixeli de-a lungul axei y (înălțime).
  - ex: 800 px * 600 px
- Număr de pixeli
  - Numărul de pixeli este numărul total de pixeli dintr-o matrice raster. Pentru a determina numărul de pixeli, se înmulțesc dimensiunile orizontale și verticale ale imaginii.
  - ex: o imagine cu dimensiunea de 30 * 18 pixeli conține 540 pixeli.

Caracteristici

- Densitatea pixelilor
  - Exprimăm densitatea pixelilor sau rezoluția unei imagini raster în pixeli pe inch (ppi) - aceasta reprezintă pixeli pe inch liniar (transversal sau în jos), nu inch pătrat..
  - Multe monitoare și televizare u o rezoluție de afișare de 72 sau 96 ppi.
  - Rezoluția determină dimensiunea maximă la care o imagine poate fi imprimată. Pentru a produce o imprimare de înaltă calitate de orice dimensiune, fotografiile digitale au nevoie de o densitate de pixeli de cel puțin 300 ppi - adică 90.000 pixeli într-un inch pătrat

Caracteristici

- Adâncimea culorii (rezoluția culorii)
  - măsură a numărului de culori diferite care pot fi reprezentate de un pixel individual. Numărul de biți alocați fiecărui pixel sau adâncimea de biți determină rezoluția culorii.
  - 8-biți (256 culori), 24 biți (16.8 millioane de culori)
  - Un desen folosind 16 culori distincte, de exemplu, ar necesita doar 4 biți pe pixel. Utilizarea unui cod cu o adâncime de biți mai mare produce un fișier mai mare, fără creșterea calității.

Avantaje și dezavantaje

- Avantaje:
  - excelent atunci când se creează imagini bogate și detaliate. Fiecare pixel dintr-o imagine raster poate avea o culoare diferită, prin urmare pot fi create imagini complexe cu orice fel de schimbări de culoare și variații.

Avantaje și dezavantaje

- Dezavantaje:
  - fișierele sunt adesea **destul de mari**, deoarece conțin toate informațiile pentru fiecare pixel al imaginii.
  - dacă este mărită, o imagine raster va arăta granulată și distorsionată, deoarece imaginile raster sunt create cu un număr finit de pixeli. La mărirea imaginii nu există informație pentru pixelii adăugați suplimentar. Software-ul de editare a fotografiilor va încerca să umple acești pixeli cât mai bine, cu toate acestea, imaginea rezultată va fi adesea neclară.
  - nu sunt furnizate informații cu privire la formele care alcătuiesc imaginea.

Avantaje și dezavantaje

Activitate practică:

1. Salvați imaginile de mai jos și analizați modul în care reacționează la modificarea dimensiunilor.
2. Care dintre cele două imagini este de tip raster?

Desenare elemente grafice

Exemplu: Trasarea unei linii presupune colorarea celulelor matricei plecând de la o ecuație matematică

Exemplu: segment de dreaptă (x1 , y1)→(x2 , y2)

Formate

- În timpul lucrului cu un fișier imagine, acesta va fi în general salvat într-un format care acceptă straturi (*.PSD, *TIFF) pentru a putea fi modificat cu ușurință.
- Pentru utilizarea imaginii într-un proiect multimedia starurile vor fi **combinate**, iar imagina va fi comprimată.
- În cazul formatele de compresie cu pierderi există întotdeauna un compromis între calitatea imaginii și dimensiunea fișierului.
- Aplicații web: https://www.photopea.com, https://pixlr.com/x/

Formate

Utilizare straturi (en: layers) în Paint.NET

Formate

- BMP (Microsoft Windows Bitmap) - *.bmp
  - cunoscut și sub numele de fișier de imagine bitmap sau format de fișier bitmap independent de dispozitiv (DIB) sau pur și simplu bitmap
  - fie necomprimat, fie comprimat folosind RLE (format de compresie fără pierderi)
  - monocrom sau color (sunt acceptate diferite adâncimi de culoare)

Formate

1. Desenați în Microsoft Paint o formă geometrică simplă (ex: un dreptunghi roșu)

2. Comparați dimensiunile fișierelor obținute prin selectarea celor patru adâncimi de culoare disponibile.

Formate

- **GIF** (Graphics Interchange Format) - ***.gif**
  - maxim **256 de culori** și transparență (pixeli transparenți);
  - **format de compresie fără pierderi**.
  - utilizat în mod obișnuit pentru sigle și alte imagini cu linii și blocuri de culoare solide.
  - permite rularea de **animații**.

Formate

https://dl.dropboxusercontent.com/u/70618257/HTML5Multimedia/giphy.gif

Formate

- JPEG (Joint Photographic Experts Group) - *.jpeg
  - permite afișarea a 16.8 milioane de culori
  - nu suportă transparență (nu permite definirea de pixeli transparenți).
  - format de compresie cu pierderi
  - este folosit cel mai adesea pentru fotografii

Formate

JPEG - o fotografie a unei pisici cu rata de compresie în scădere și, prin urmare, creșterea calității, de la stânga la dreapta[1].

Formate

Photoshop – meniul Save for Web

Formate

- **PNG** (Portable Network Graphics) - ***.png**
  - cel mai utilizat format de compresie a imaginilor fără pierderi de pe Internet [1]
  - creat ca un înlocuitor îmbunătățit, nebrevetat, pentru formatul GIF [1]
  - permite afișarea a 16.8 million colors; suportă transparență; permite utilizarea unui număr mai mic de culori pentru a reduce dimensiunea fișierului (PNG 8, sau PNG cu reprezentarea culorilor pe 8-biți) [2]
  - format de compresie fără pierderi[2]
  - utilizat pentru o gamă largă de imagini, inclusiv *favicons* (the small web page icons in browser tabs) [2]
  - fișierele PNG pot fi foarte mici, dar pentru fotografii cu multe culori, pot avea o dimensiune mai mare decât fișierele JPEG la o calitate comparabilă [2]

Formate

Imagine compusă care compară compresia cu pierderi în JPEG cu compresia fără pierderi în PNG: artefactele JPEG sunt ușor vizibile în fundal, unde imaginea PNG are o culoare solidă [1].

Formate

- WebP
  - acceptă compresia cu pierderi prin codificare predictivă bazată pe codecul video VP8 și compresia fără pierderi care folosește substituții pentru repetarea datelor.
  - codificarea predictivă folosește valorile din blocurile vecine de pixeli pentru a prezice valorile dintr-un bloc și apoi codifică doar diferența.
  - imaginile WebP cu pierderi sunt în medie cu 25–35% mai mici decât imaginile JPEG cu niveluri de compresie similare vizual. Imaginile WebP fără pierderi sunt de obicei cu 26% mai mici decât aceleași imagini în format PNG.
  - permite redarea de animații
  - detalii: https://developers.google.com/speed/webp , https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Image_types#WebP

Formate

- Other formats:
  - ICO - Icon Resource File;
  - DIB - Device Independent Bitmap;
  - PCX - PC PaintBrush File Format;
  - **TIFF** - Tag Image File Format.

Opțional

Imagini de dimensiuni mari

- Dubai: http://gigapan.com/gigapans/48492
- Size: 44.88 Gigapixels

Compresie

- Algoritmi de compresie:
  - Run-length encoding (RLE)
  - Lempel–Ziv–Welch (LZW)
  - Huffman
  - JPEG

Compression - Run Length Encoding - RLE

- one of the simpler strategies to achieve **lossless** compression
- can be used to compress bitmapped image files (ex: *pcx format). Bitmapped images can easily become very large because each pixel is represented with a series of bits that provide information about its color.
- RLE generates a code to “flag” the beginning of a line of pixels of the same color. That color information is then recorded just once for each pixel. In effect, RLE tells the computer to repeat a color for a given number of adjacent pixels rather than repeating the same information for each pixel over and over. The RLE compressed file will be smaller, but it will retain all the original image data—it is “lossless.”

Compression - Run Length Encoding - RLE

Compression - Lempel–Ziv–Welch (LZW)

- **lossless** data compression algorithm
- used in the GIF image format

- Encoding
  - Initialize the dictionary to contain all strings of length one.
  - Find the longest string W in the dictionary that matches the current input.
  - Emit the dictionary index for W to output and remove W from the input.
  - Add W followed by the next symbol in the input to the dictionary.
  - Go to Step 2.

JPEG Compression

- **lossy** data compression algorithm

- Steps
  1. color space transformation
  2. downsampling
  3. block splitting
  4. transformation
  5. quantization
  6. encoding

JPEG Compression - Step 1 - Color space transformation

- The representation of the colors in the image is converted from **RGB** to **Y'CBCR**, consisting of one luma component (Y'), representing brightness, and two chroma components, (CB and CR), representing color. This step is sometimes skipped.

JPEG Compression - Step 1 - Color space transformation

- The $Y'C_BC_R$ color space conversion allows greater compression without a significant effect on perceptual image quality (or greater perceptual image quality for the same compression).

- The compression is more efficient because the brightness information, which is more important to the eventual perceptual quality of the image, is confined to a single channel. This more closely corresponds to the perception of color in the human visual system.

JPEG Compression – Step 2 - Downsampling

- Due to the densities of color- and brightness-sensitive receptors in the human eye, humans can see considerably more fine detail in the brightness of an image (the Y' component) than in the hue and color saturation of an image (the Cb and Cr components). Using this knowledge, encoders can be designed to compress images more efficiently.

- The transformation into the $Y'C_BC_R$ color model enables the next usual step, which is to reduce the spatial resolution of the Cb and Cr components.

JPEG Compression – Step 2 - Downsampling

JPEG Compression – Step 2 - Downsampling

- The ratios at which the downsampling is ordinarily done for JPEG images are:
  - 4:4:4 - no downsampling,
  - 4:2:2 - reduction by a factor of 2 in the horizontal direction
  - 4:2:0 - reduction by a factor of 2 in both the horizontal and vertical directions (most common).

- For the rest of the compression process, Y', Cb and Cr are processed separately and in a very similar manner.

JPEG Compression – Step 3 - Block splitting

- Each channel must be split into 8×8 blocks

- Depending on chroma subsampling, this yields Minimum Coded Unit (MCU) blocks of size 8×8 (4:4:4 – no subsampling), 16×8 (4:2:2), or most commonly 16×16 (4:2:0).

JPEG Compression – Step 4 - Discrete cosine transform

- Each channel must be split into blocks of size 8 x 8 pixels

- If we have chosen an image whose dimensions are 160 x 240 = 8*20 x 8*30. So this step creates 20 x 30 = 600 blocks.

JPEG Compression – Step 4 - Discrete cosine transform

5	176	193	168	168	170	167	165
6	176	158	172	162	177	168	151
5	167	172	232	158	61	145	214
33	179	169	174	5	5	135	178
8	104	180	178	172	197	188	169
63	5	102	101	160	142	133	139
51	47	63	5	180	191	165	5
49	53	43	5	184	170	168	74

JPEG Compression – Step 4 - Discrete cosine transform

- Before computing the DCT of the 8×8 block, its values are shifted from a positive range to one centered on zero.

- We subtract 127 from each pixel intensity in each block. This step centers the intensities about the value 0 and it is done to simple the mathematics of the transformation and quantization steps.

JPEG Compression – Step 4 - Discrete cosine transform

5	176	193	168	168	170	167	165
6	176	158	172	162	177	168	151
5	167	172	232	158	61	145	214
33	179	169	174	5	5	135	178
8	104	180	178	172	197	188	169
63	5	102	101	160	142	133	139
51	47	63	5	180	191	165	5
49	53	43	5	184	170	168	74

-122	49	66	41	41	43	40	38
-121	49	31	45	35	50	41	24
-122	40	45	105	31	-66	18	87
-94	52	42	47	-122	-122	8	51
-119	-23	53	51	45	70	61	42
-64	-122	-25	-26	33	15	6	12
-76	-80	-64	-122	53	64	38	-122
-78	-74	-84	-122	57	43	41	-53

JPEG Compression – Step 4 - Discrete cosine transform

- The JPEG Image Compression Standard relies on the *Discrete Cosine Transformation (DCT)* to transform the image. The DCT is applied to each 8 x 8 block.

- The DCT is a product C = U B U^T where B is an 8 x 8 block from the preprocessed image and U is a special 8 x 8 matrix.

- Loosely speaking, the DCT tends to push most of the high intensity information (larger values) in the 8 x 8 block to the upper left-hand of C with the remaining values in C taking on relatively small values.

JPEG Compression – Step 4 - Discrete cosine transform

-27.500	-213.468	-149.608	-95.281	-103.750	-46.946	-58.717	27.226
168.229	51.611	-21.544	-239.520	-8.238	-24.495	-52.657	-96.621
-27.198	-31.236	-32.278	173.389	-51.141	-56.942	4.002	49.143
30.184	-43.070	-50.473	67.134	-14.115	11.139	71.010	18.039
19.500	8.460	33.589	-53.113	-36.750	2.918	-5.795	-18.387
-70.593	66.878	47.441	-32.614	-8.195	18.132	-22.994	6.631
12.078	-19.127	6.252	-55.157	85.586	-0.603	8.028	11.212
71.152	-38.373	-75.924	29.294	-16.451	-23.436	-4.213	15.624

JPEG Compression – Step 5 - Quantization

- elements near zero will converted to zero and other elements will be shrunk so that their values are closer to zero. All quantized values will then be rounded to integers.

- quantization is performed in order to obtain integer values and to convert a large number of the values to 0. The Huffman coding algorithm will be much more effective with quantized data and the hope is that when we view the compressed image, we haven't given up too much resolution.

JPEG Compression – Step 5 - Quantization

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

Note that the values are largest in the lower right corner of Z. These divides should produce values close to zero so that the rounding function will convert the quotient to zero

JPEG Compression – Step 5 - Quantization

-2	-19	-15	-6	-4	-1	-1	0
14	4	-2	-13	0	0	-1	-2
-2	-2	-2	7	-1	-1	0	1
2	-3	-2	2	0	0	1	0
1	0	1	-1	-1	0	0	0
-3	2	1	-1	0	0	0	0
0	0	0	-1	1	0	0	0
1	0	-1	0	0	0	0	0

JPEG Compression – Step 6 - Encoding

It involves arranging the image components in a "zigzag" order employing run-length encoding (**RLE**) algorithm that groups similar frequencies together, inserting length coding zeros, and then using **Huffman** coding on what is left.

JPEG Compression – Step 5 - Quantization

- quantization makes the JPEG algorithm an example of *lossy compression*:
  - the DCT step is completely invertible - that is, we applied the DCT to each block B by computing C = U B U^T. It turns out we can recover B by the computation B = U^T C U.
  - converting small values to 0 and rounding all quantized values are not reversable steps. The ability to recover the original image is lost.

HTML5 Image Element

- afișarea imaginilor în HTML :

```
<img alt="" src="" title="">
```

- tipuri de imagini raster suportate: JPEG, GIF, PNG, WEBP
- Element: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img
- API: https://developer.mozilla.org/en-US/docs/Web/API/HTMLImageElement

HTML5 Image Element

- Proprietăți:
  - width, height: dimensiunea obiectului în fereastră;
  - naturalWidth, naturalHeight: dimensiunea matricei raster;
  - src: URL-ul imaginii sursă; modificarea determină începutul procesului de încărcare (asincron).

- Evenimente:
  - load: declanșat când o resursă și resursele dependente ale acesteia s-au terminat de încărcat.

HTML5 Image Element

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

```
//Vanilla JavaScript
let image = document.createElement('img');
image.addEventListener('load', function(e){
    //document.body.appendChild(e.target)
    document.body.appendChild(image)
});
image.src = "mask.png";
// echivalent cu image.setAttribute('src', "mask.png");
```

HTML5 Image Element

Rescrieți codul din slide-ul anterior utilizând JavaScript “Promise”.

HTML5 Canvas Element / HTMLCanvasElement

- Poate fi folosit pentru a desena grafice, a face compoziții foto sau chiar a efectua animații.

- Declararea unui element de tip canvas :

```
<canvas id="test" style="width:250px; height:150px">
    An alternative text describing what your canvas displays.
</canvas>
```

- Element: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas

- API: https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement

HTML5 Canvas Element - Metode

- getContext(contextType, contextAttributes); - returnează contextul grafic de desenare al canvas-ului sau null, dacă contextul solicitat nu este suportat;

```
const canvas = document.getElementById('test');  // sau const canvas = $('test')[0];
const w = canvas.width, h = canvas.height;
const ctx = canvas.getContext('2d');
```

- contextType: specifică tipul de context grafic utilizat pentru desenarea pe canvas.
- contextAttributes: setări suplimentare
- docs: developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement/getContext

HTML5 Canvas Element - Metode

- valori posibile contextType:
  - "2d" - determină returnarea unui obiect de tipul CanvasRenderingContext2D reprezentând un context de desenare bidimensională.
  - "webgl" / "webgl2" - determină returnarea unui obiect de tipul WebGLRenderingContext reprezentând un context de desenare tridemensională. Acest tip de context grafic este disponibil doar în browser-ele care implementează WebGL version 1 (OpenGL ES 2.0) / 2 (OpenGL ES 3.0).
  - "bitmaprenderer" - determină returnarea unui obiect de tip ImageBitmapRenderingContext care oferă doar funcționalitatea necesară pentru a înlocui conținutul canvas-ului cu un ImageBitmap.

HTML5 Canvas Element - Metode

1. Verificați tipul de element HTML utilizat de maps.google.com pentru afișarea hărții.

2. Care credeți că este tipul de context grafic utilizat?

HTML5 Canvas Element - Interfața CanvasRenderingContext2D

- Utilizată pentru desenarea de formelor geometrice, imaginilor, textelor și a altor elemente pe <canvas>.

- API: developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D

- Desenarea de dreptunghiuri:
  - clearRect(x, y, width, height) - ștergere suprafață
  - fillRect(x, y, width, height) - umplere suprafață
  - strokeRect(x, y, width, height) - desenare contur

HTML5 Canvas Element - Interfața CanvasRenderingContext2D

- Desenarea textului:
  - fillText(text, x, y [, maxWidth]) – desenare text (cu umplere);
  - strokeText(text, x, y [, maxWidth]) – desenare text (doar contur);
  - measureText(text) - returns a TextMetrics object (include o proprietate width).

- Desenare folosind căi:
  - documentație

- Stiluri pentru linii:
  - documentație

HTML5 Canvas Element - Interfața CanvasRenderingContext2D

- Stiluri pentru text:
  - check documentation
- Stiluri pentru umplere și contur (en: Fill and stroke styles):
  - check documentation
- Gradient și pattern:
  - check documentation
- Umbre:
  - check documentation

HTML5 Canvas Element - Interfața CanvasRenderingContext2D

- Desenarea imaginilor:
  - fără scalare
    - void ctx.drawImage(image, dx, dy);
  - cu scalare
    - void ctx.drawImage(image, dx, dy, dWidth, dHeight);
    - void ctx.drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight);
  - API: Mozilla Developer Network

Salvarea și restabilirea setărilor

- CanvasRenderingContext2D.**save()**
  - salvează setările de desenare prin adăugarea acestora la o stivă
- CanvasRenderingContext2D.**restore()**
  - modifică setările de desenare prin extragerea primei intrări din stiva utilizată pentru salvarea setărilor

Salvarea și restabilirea setărilor

- Setări salvate:
  - matricea curentă de transformare.
  - zona curentă de desenare (en: clipping region).
  - setările pentru linii punctate (en: dash list).
  - atributele: strokeStyle, fillStyle, globalAlpha, lineWidth, lineCap, lineJoin, miterLimit, lineDashOffset, shadowOffsetX, shadowOffsetY, shadowBlur, shadowColor, globalCompositeOperation, font, textAlign, textBaseline, direction, imageSmoothingEnabled.

Salvarea și restabilirea setărilor

- 1. Exemplu save și restore
- 2. Exemplu save și restore în contextul librăriei BarChart

HTML5 Canvas Element - Transformări

HTML5 Canvas Element - CanvasRenderingContext2D Interface

- Translație
  - translate(x, y)
    - modifică originea.
    - x indică deplasarea pe orizontală și y indică deplasarea pe verticală.
  - API: Mozilla Developer Network

HTML5 Canvas Element - CanvasRenderingContext2D Interface

```
function draw() {
  const ctx = document.getElementById('canvas').getContext('2d');
  for (let i=0;i<3;i++) {
    for (let j=0;j<3;j++) {
      ctx.save();
      ctx.fillStyle = 'rgb('+(51*i)+','+(255-51*i)+',255)';
      ctx.translate(10+j*50,10+i*50);
      ctx.fillRect(0,0,25,25);
      ctx.restore();
    }
  }
}
```

- Fără apelarea metodei translate() toate dreptunghiurile ar fi desenate (0,0). Utilizând metoda translate() nu mai este necesară modificarea parametrilor metodei fillRect().
- JSFiddle: https://jsfiddle.net/liviucotfas/9tfdegn7/

HTML5 Canvas Element - CanvasRenderingContext2D Interface

- Rotație
  - rotate(unghi)
    - rotește sistemul de coordonate cu unghiul specificat (în radiani)
    - rotația se efectuează față de originea canvas-ului, cu excepția cazului în care aceasta a fost modificată cu ajutorul metodei translate
    - Notă: Unghiul este exprimat în radiani, nu în grade. Conversie: radians = (Math.PI/180)*degrees.
    - API: Mozilla Developer Network

HTML5 Canvas Element - CanvasRenderingContext2D Interface

- dreptunghiul 1: rotit față de originea elementului de tip <canvas>
- dreptunghiul 2: rotit față de centrul dreptunghiului cu ajutorul metodei **translate()**.
- JSFiddle: https://jsfiddle.net/liviucotfas/2yf241q1/

HTML5 Canvas Element - CanvasRenderingContext2D Interface

- Scalare
  - scale(x, y)
    - Modifică dimensiunea unității de desenare pe canvas pe orizontală sau pe verticală.
    - Ambii parametri sunt numere reale. Valorile mai mici de 1,0 reduc dimensiunea unității de desenare, iar valorile peste 1,0 măresc dimensiunea unității de desenare. Valorile de 1,0 lasă unitatea de aceeași dimensiune.
    - Utilizând valori negative putem desena în “oglindă”.
    - API: Mozilla Developer Network

HTML5 Canvas Element - CanvasRenderingContext2D Interface

- rectangle: scaled
- text: mirrored.
- JSFiddle: https://jsfiddle.net/liviucotfas/Lyycq7gv/

Game Development

Question 1: SVG or Canvas?

Question 2: Context de desenare utilizat? (webgl, webgl2, 2d)?

https://techcrunch.com/2020/10/29/google-brings-halloween-to-life-using-augmented-reality/

https://www.google.com/doodles/halloween-2020

HTML5 Canvas Element - CanvasRenderingContext2D Interface

- Combinare transformări
  - browser-ul utilizează o matrice de transformare

```
context.scale(-.5, 1);
context.rotate(45 * Math.PI / 180);
context.translate(40, 10);
```

  - translația, rotația și scalarea modifică valorile din această matrice

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

HTML5 Canvas Element - CanvasRenderingContext2D Interface

- Transformări
  - permit modificări directe ale matricii de transformare
  - transform(m11, m12, m21, m22, dx, dy)
    - înmulțește matricea de transformare curentă cu matricea descrisă de argumentele sale.
    - API: Mozilla Developer Network

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

HTML5 Canvas Element - CanvasRenderingContext2D Interface

- m11 - Horizontal scaling.
- m12 - Horizontal skewing.
- m21 - Vertical skewing.
- m22 - Vertical scaling.
- dx - Horizontal moving.
- dy - Vertical moving.

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

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

HTML5 Canvas Element - CanvasRenderingContext2D Interface

- ex: transformare coordonate

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

HTML5 Canvas Element - CanvasRenderingContext2D Interface

1. Utilizați metoda “transform” pentru a scrie codul echivalent cu “context.translate(50, 50);” Exemplificați cerința cu ajutorul unui dreptunghi:

2. Utilizați metoda “transform” pentru a scrie codul echivalent cu “context.scale(1, -1);”

HTML5 Canvas Element - CanvasRenderingContext2D Interface

- resetTransform()
  - reinițializează matricea de transformare curentă. Este echivalentă cu apelul: `ctx.setTransform(1, 0, 0, 1, 0, 0)`;

- setTransform(m11, m12, m21, m22, dx, dy)
  - reinițializează matricea de transformare curentă și apoi apelează metoda transform() cu argumentele specificate.
  - echivalentă cu apelarea metodelor `*resetTransform()*` și `*transform()*`

HTML5 Canvas Element - ImageData Interface

- Interfața ImageData oferă acces la pixelii care compun o anumită zonă din elementul de tip <canvas>.
- API: Mozilla Developer Network
- Putem obține o instanță de tip ImageData:
  - new ImageData();
  - CanvasRenderingContext2D.createImageData();
  - CanvasRenderingContext2D.getImageData();

HTML5 Canvas Element - ImageData Interface

- new ImageData(): utilă în context webworker
- CanvasRenderingContext2D.**createImageData(sw, sh)**: creează un obiect ImageData nou, gol, cu dimensiunile specificate. Toți pixelii din noul obiect sunt negri transparenți.
- CanvasRenderingContext2D.**createImageData(imageData)**: creează un obiect ImageData nou, gol, cu dimensiunile obiectului imageData primit ca parametru. Toți pixelii din noul obiect sunt negri transparenți.
- CanvasRenderingContext2D.**getImageData(sx, sy, sw, sh)**: extrage o porțiune din imagine ca obiect ImageData

HTML5 Canvas Element - ImageData Interface

- CanvasRenderingContext2D.putImageData(imagedata, dx, dy): aplică datele pe imaginea afișată în Canvas

HTML5 Canvas Element - ImageData Interface

- Utilizați metodele getImageData și putImageData pentru a copia o porțiune dintr-un canvas.

HTML5 Canvas Element - ImageData Interface

- Proprietăți
  - ImageData.data : Uint8ClampedArray = vector care conține informațiile cu privire la culoarea pixelilor în ordinea RGBA, cu valori între 0 și 255.

HTML5 Canvas Element - ImageData Interface

[255,0,0,255, 0,0,0,255, 255,255,255,255, 197,52,144,255]

HTML5 Canvas Element - ImageData Interface

1. Creați o imagine similară cu cea de pe slide-ul anterior.
2. Verificați rezultatul întors de metoda getImageData.
3. Implementați o aplicație de tip Color Picker.

HTML5 Canvas Element - ImageData Interface

- ImageData.width - unsigned long reprezentând lățimea reală, în pixeli, a ImageData.
- ImageData.height - unsigned long reprezentând înălțimea reală, în pixeli, a ImageData.

HTML5 Canvas Element - ImageData Interface

- Parcurgerea tuturor pixelilor din canvas:

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

for (let i = 0; i < data.length; i+=4) {
    const red = imageData.data[i]; // [0..255]
    const green = imageData.data[i+1]; // [0..255]
    const blue = imageData.data[i+2]; // [0..255]
    const transparency = imageData.data[i+3]; // [0..255]
}
```

HTML5 Canvas Element - ImageData Interface

- Parcurgerea tuturor pixelilor din canvas:

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

for (let y = 0; y < canvas.height; y++) { // linii
   for (let x = 0; x < canvas.width; x++) { //coloanele

      const i = (y * canvas.width * 4) + x * 4;

      const red =data[i]; // [0..255]
      const green =data[i+1]; // [0..255]
      const blue =data[i+2]; // [0..255]
      const transparency =data[i+3]; // [0..255]
   }
}
```

HTML5 Canvas Element - ImageData Interface

- Pornind de la codul prezentat în slide-ul anterior scrieți o funcție care transformă în tonuri de gri doar jumătatea stângă a unei imagini.

HTML5 Canvas Element

- Examples:
  - Color picker: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Pixel_manipulation_with_canvas#A_color_picker
  - Zoom: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Pixel_manipulation_with_canvas#zooming_and_anti-aliasing

HTML5 Canvas Element

- Exemple:
  - http://www.williammalone.com/articles/html5-canvas-javascript-paint-bucket-tool/
  - http://www.williammalone.com/articles/create-html5-canvas-javascript-drawing-app/

Efecte

- Implementați utilizând metodele ***getImageData*** și ***putImageData*** efectele de pe slide-urile următoare.

Efecte

- Color Filter
  - Red filter: $r' = r$; $g' = 0$; $b' = 0$
- Negative / Invert
  - $r' = 255 - r$; $g' = 255 - g$; $b' = 255 - b$;
- Tonuri de gri / Grayscale
  - $r' = g' = b' = 0.299 * r + 0.587 * g + 0.114 * b$
  - $r' = g' = b' = (r + g + b) / 3$

Efecte

- Threshold
  - v = (0.2126*r + 0.7152*g + 0.0722*b >= threshold) ? 255 : 0; r′= g′ = b′ = v

- Sepia
  - r' = (r * .393) + (g *.769) + (b * .189)
  - g' = (r * .349) + (g *.686) + (b * .168)
  - b' = (r * .272) + (g *.534) + (b * .131)

Efecte

- Luminozitate / Brightness
  - $r' = r + value$; if $(r' > 255)$ $r' = 255$ else if $(r' < 0)$ $r' = 0$;
  - $g' = g + value$; if $(g' > 255)$ $g' = 255$ else if $(g' < 0)$ $g' = 0$;
  - $b' = b + value$; if $(b' > 255)$ $b' = 255$ else if $(b' < 0)$ $b' = 0$;

Efecte

- Example:
  - https://www.html5rocks.com/en/tutorials/canvas/imagefilters/#toc-simplefilters

- Bibliogarfie suplimentară / exemple:
  - https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Pixel_manipulation_with_canvas

Efecte – Filtre de convoluție / Convolution filters

- Permit implementarea de filtre generice pentru procesarea imaginilor: blur, sharpen, emboss, edge detection.

- Calculează noua valoare pentru un pixel pe baza valorilor pixelilor adiacenți din imaginea originală.

- **Examplu de filtru:** blur filter

$$\begin{bmatrix} 1/9, & 1/9, & 1/9, \\ 1/9, & 1/9, & 1/9, \\ 1/9, & 1/9, & 1/9 \end{bmatrix}$$

- **Notă**: pentru a menține luminozitatea imaginii, suma valorilor din matrice ar trebui să fie egală cu unu.

Efecte – Filtre de convoluție / Convolution filters

- Live example:
  - https://setosa.io/ev/image-kernels

Efecte – Filtre de convoluție / Convolution filters

- Alte exemple de matrici de convoluție: https://en.wikipedia.org/wiki/Kernel_(image_processing)

WebGL

- Documentation: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API

- Three.js
  - cross-browser JavaScript library and Application Programming Interface (API) used to create and display animated 3D computer graphics in a web browser
  - Documentation: https://threejs.org/
  - Example: https://heraclosgame.com/

WebGL

- Babylonjs
  - JavaScript framework for building 3D games and experiences with HTML5, WebGL, WebVR and Web Audio
  - Documentation: http://babylonjs.com/

Grafică vectorială

Grafică vectorială

- În cazul **graficii raster**, imaginea este formată din pixeli dispuți pe rânduri și coloane, sub forma unei matrici.

- În cazul **graficii vectoriale** computerului i se oferă un **set de comenzi** pe care le execută pentru a desena imaginea. Imaginile conțin căi (en: paths) compuse din puncte, linii, curbe și forme (en: shapes).

```
<svg viewBox="0 0 220 100" xmlns="http://www.w3.org/2000/svg">
        <!-- Simple rectangle -->
        <rect width="100" height="100" />
        <!-- Rounded corner rectangle -->
        <rect x="120" width="100" height="100" rx="15" />
</svg>
```

Grafică vectorială

- Fiecare formă descrisă vectorială definește conturul unei regiuni geometrice care conține informații despre culoare.

```
<svg viewBox="0 0 220 100"
xmlns="http://www.w3.org/2000/svg">
        <!-- Simple rectangle -->
        <rect width="100" height="100" />
        <!-- Rounded corner rectangle -->
        <rect x="120" width="100" height="100" rx="15" />
</svg>
```

Grafică vectorială

Grafică vectorială

- Deoarece căile (en: paths) pot fi redimensionate matematic, imaginile vectoriale pot fi redimensionate, fără a se pierde din claritatea imaginii.

```
<svg viewBox="0 0 220 100"
xmlns="http://www.w3.org/2000/svg">
        <!-- Simple rectangle -->
        <rect width="100" height="100" />
        <!-- Rounded corner rectangle -->
        <rect x="120" width="100" height="100" rx="15" />
</svg>
```

Redimensionare / Zoom

Redimensionare / Zoom

Animație

- Grafică vectorială poate fi utilizată pentru crearea de animații. În loc să desenăm fiecare cadru al proiectului separat (24 de cadre care apar în fiecare secundă) se pot utiliza instrumente software de animație scare interpolează matematic pozițiile componentelor în cadrele intermediare.

Avantaje și dezavantaje

- Avantaje :
  - Pentru imagini relativ simple, lista comenzilor utilizate pentru desenare ocupă mult mai puțin spațiu în fișier decât o versiune raster a aceleași imagini.
  - Un program de desenare poate utiliza o comandă similară cu “RECT 300, 300, RED” pentru a desena un pătrat roșu cu laturile de 300 pixeli. Fișierul pentru această imagine conține 15 octeți care codifică informațiile alfanumerice din comandă.
  - Aceeași imagine ar putea fi creată și cu un program de grafică raster. Utilizând o adâncime de culoare de 8-biți (un byte per pixel), fișierul ar avea o dimensiune necomprimată de 90,000 bytes (300 × 300). Dimensiunile mult mai mici ale fișierelor vectoriale pot oferi beneficii semnificative în cazul aplicațiilor multimedia.

Avantaje și dezavantaje

- Imaginile vectoriale pot fi **redimensionate** fără pierderea calității. Imaginile vectoriale sunt mărite modificând parametrii formelor componente. Noua imagine poate fi apoi redesenată cu precizie la dimensiuni mai mari, fără distorsiunile tipice ale graficelor bitmap-uri mărite.

- Imaginile care sunt necesare în mai multe dimensiuni, cum ar fi sigla companiei, sunt adesea reprezentate cel mai bine cu ajutorul graficii vectoriale.

1. Salvați pe dispozitivul vostru logo-ul alăturat și deschideți-l cu ajutorul unui editor text (e.g., Notepad, VS Code).
2. Accesați pagina Facebook și alegeți opțiunea Inspect pentru a analiza codul aferent logo-ului.

Avantaje și dezavantaje

- Dezavantaje:
  - Principalul dezavantaj al graficii desenate vectorial este lipsa controlului asupra pixelilor individuali ai imaginii. Prin urmare grafica vectorială nu este potrivită pentru afișarea și editarea imaginilor foto-realiste.

Formate

- SVG (Scalable Vector Graphics)
  - utilizează formatul XML pentru imagini bidimensionale
  - oferă suport pentru interactivitate și animație

- DXF (Drawing Exchange Format)
  - este un format CAD dezvoltat de Autodesk pentru a permite interoperabilitatea între AutoCAD și alte programe.

Formate

- EPS (Encapsulated Post Script)
  - creat de Adobe Systems pentru reprezentarea graficii vectoriale
  - utilizează un limbaj numit Post Script

- SHP (Shapefile)
  - dezvoltat de Esri pentru reprezentarea informațiilor geografice în contextul aplicațiilor GIS

Formate

Harta vectorială simplă:

- puncte (foraje),
- polilinii (râuri),
- poligoane (lac).

SVG – Scalable Vector Graphics

- documentație: https://developer.mozilla.org/en-US/docs/Web/SVG

- format de imagine vectorială bazat pe XML pentru grafică bidimensională

```
<svg viewBox="0 0 400 180" xmlns="http://www.w3.org/2000/svg">
 <rect
  x="50" y="20"
  width="150" height="150"
  fill="red" stroke="black"
  stroke-width="5"
  opacity="0.5" />
</svg>
```

- oferă suport pentru interactivitate și animație

SVG – Scalable Vector Graphics

SVG – Scalable Vector Graphics

- Crearea imaginilor SVG
  - imaginile SVG pot fi create prin intermediul oricărui editor text, dar sunt mai ușor de construit cu ajutorul programelor de grafică vectorială precum Adobe Illustrator sau Inkscape.

SVG – Scalable Vector Graphics

- Linie

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

- Example:
  - http://www.w3schools.com/graphics/svg_line.asp

```
<svg height="210" width="500">
        <line x1="0" y1="0" x2="200" y2="200" style="stroke:rgb(255,0,0);stroke-width:2" />
</svg>
```

SVG – Scalable Vector Graphics

- Dreptunghi

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

- *Example*
  - *http://www.w3schools.com/graphics/svg_rect.asp*

```
<svg width="400" height="110">
 <rect width="300" height="100" style="fill:rgb(0,0,255);stroke-width:3;stroke:rgb(0,0,0)" />
</svg>
```

SVG – Scalable Vector Graphics

- Cerc

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

- *Example*
  - *http://www.w3schools.com/graphics/svg_circle.asp*

```
<svg height="100" width="100">
 <circle cx="50" cy="50" r="40" stroke="black" stroke-width="3" fill="red" />
</svg>
```

SVG – Scalable Vector Graphics

- Elipsă

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

- Poligon

<polygon points="x1,y1 x2,y2 ..."/>

- Polilinie

<polyline points="x1,y1 x2,y2 ..."/>

- Text

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

SVG – Scalable Vector Graphics

- Grupare elemene

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

- Definire grupuri

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

- Reutilizare grupuri

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

Interacțiunea cu SVG utilizând CSS

- Documentație: https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started/SVG_and_CSS

- Atribute de bază
  - stroke – culoare linie
  - stroke-opacity – opacitate linie
  - stroke-width – dimensiune linie

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

Interacțiune cu SVG utilizând JavaScript

- similar cu abordarea utilizată pentru elementele HTML;
- particularități:
  - atunci când creăm un element, trebuie să folosim spațiul de nume (en: namespace) SVG.

```
document.createElementNS("http://www.w3.org/2000/svg", “TAG_SVG")
```

SVG – Scalable Vector Graphics

- Examplu jQuery

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

Audio

Cuprins

1. Noțiuni generale

2. Numerizarea sunetului

3. Formate audio

4. Compresia sunetului

5. Sunetul în context Web

Noțiuni generale

- **Sunetul** este o vibrație propagată printr-un mediu material sub forma unei unde mecanice.
- Mediul este de obicei aer, deși sunetul poate fi transmis și prin solide și lichide.
- Exemplu: bătutul din palme produce un sunet prin comprimarea și deplasarea bruscă a moleculelor de aer. Perturbarea este transmisă moleculelor adiacente și se propagă prin spațiu sub forma unei unde. Auzim aplauzele mâinilor atunci când aceste vibrații ajung la urechilor noastre.

Audio

- Undele sonore sunt adesea comparate cu undele produse atunci când o piatră este aruncată într-un iaz. Piatra deplasează apa generând puncte înalte sau vârfuri de undă și puncte joase sau jgheaburi. Vedem aceste vârfuri și jgheaburi alternante ca modele de valuri care se deplasează spre exterior din punctul de impact al pietrei. Bătutul din plame produce, în mod similar, variații de presiune în aer. Aceste modificări ale presiunii aerului produc modele de unde care se răspândesc în toate direcțiile de la sursa sunetului.

Audio

- Urechea umană poate percepe sunete cu frecvența cuprinsă între aproximativ 20 Hz și 20 kHz

- Zgomot: caz particular de sunet caracterizat prin lipsa încărcăturii informaționale

Reprezentare

Caracteristici

- amplitudine
- frecvență
- durată

Caracteristici

- **Amplitudinea** este o măsură a presiunii sonore sau a cantității de energie asociată sunetului. Aceasta este reprezentată de verticală sau axa y a undei sinusoidale. Amplitudinea este percepută ca **volum al sunetului**, care este de obicei măsurat în **decibeli** (dB).

- În general, sunetele cu amplitudini mai mari sunt mai puternice. Spectrul auzului uman este de la aproximativ **3** până la **140 dB**. Fiecare creștere de **10 dB** dublează aproximativ volumul perceput al unui sunet.

Caracteristici

- **Frecvența** este numărul de repetări ale unei forme de undă într-un anumit interval de timp. Este reprezentată pe axa orizontală ca distanță între două vârfuri de undă sau jgheaburi. Frecvența este măsurată în **hertz** (Hz).
- Un **hertz** este **o repetare** a unei forme de undă într-**o secundă** de timp.

- Frecvențele înalte produc sunete de înălțime mai mare, iar frecvențele joase produc înălțimea joasă. Oamenii pot percepe un interval de frecvență de la 20 Hz la 20.000 Hz (sau 20 kilohertz (kHz), mii de hertz), deși majoritatea adulților nu pot auzi frecvențe peste 16 kHz.

Caracteristici

- **Durata sunetului** este echivalentă cu durata în timp a sunetului (ex: măsurată în secunde).

Audio Digital

- Tehnicile digitale reprezintă sunetul într-o formă discretă (discontinuă) de informații

- Există două tipuri majore de sunet digital:
  - *eșantionat*
  - *sintetizat*.

Audio Digital

- **Sunetul sintetizat** este un sunet generat (sau sintetizat) de computer. Un fișier pentru sunetul sintetizat conține instrucțiuni pe care computerul le folosește pentru a produce sunetul.

- **Sunetul eșantionat** este de obicei utilizat pentru a capta și edita sunete naturale, cum ar fi vorbirea umană, spectacole muzicale și dramatice, ciripit de păsări, lansări de rachete și așa mai departe. **Sunetul sintetizat** este utilizat în general pentru a crea compoziții muzicale originale sau pentru a produce efecte sonore noi.

Audio Digital

- **Sunetul eșantionat** este o înregistrare digitală a undelor sonore analogice existente anterior. Un fișier pentru sunetul eșantionat conține multe mii de valori numerice, fiecare dintre acestea fiind o înregistrare a amplitudinii undei sonore la un moment particular, o **eșantionare a sunetului**.

Numerizarea sunetului

- presupune stocarea și prelucrarea sunetului în format digital

- sunetul este captat prin înregistrarea a numeroase măsurători separate ale amplitudinii unei unde utilizând un convertor ADC sau analog-digital. Un dispozitiv analogic, cum ar fi un microfon sau amplificatorul dintr-un sistem de difuzoare, generează un model de tensiune care variază continuu pentru a se potrivi cu unda sonoră originală. ADC eșantionează aceste tensiuni de mii de ori pe secundă. Aceste valori digitale sunt apoi utilizate pentru a recrea sunetul original prin conversia informațiilor digitale înapoi într-o formă analogică utilizând un convertor DAC sau digital-analog. DAC utilizează valorile amplitudinii pentru a genera variații de tensiune în curentul care alimentează difuzoarele pentru a reproduce sunetul

Numerizarea sunetului

Etape:

- Convertirea sunetului în semnal electric (micofon)
- Eșantionarea și cuantificarea semnalului (ADC)
- 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

Numerizarea sunetului

- Eșantionarea digitală înlocuiește forma de undă continuă a sunetului original cu o nouă undă creată dintr-un număr fix de măsurători discrete.
- Unele informații se pierd în cadrul procesului de eșantionare, deoarece o undă continuă este infinit divizibilă și eșantionarea produce întotdeauna un număr finit de valori.
- Calitatea sunetului eșantionat depinde de doi factori conectați direct la acest proces de eșantionare: **rezoluția de eșantionare** (en: sample resolution) și **rata de eșantionare**.

Numerizarea sunetului

- **Rezoluția de eșantionare** (en: sample resolution). Fiecare măsurare a amplitudinii efectuată de un ADC este înregistrată folosind un număr fix de biți. Numărul de biți utilizați pentru a codifica amplitudinea este cunoscut sub numele de rezoluție de eșantionare.

- Rezoluțiile de eșantionare variază între 8 și 32 de biți, cele mai frecvente fiind standardele CD-Audio pe 16 biți și standardele DVD-Audio pe 24 biți

Numerizarea sunetului

Each measurement is assigned a number (byte) according to its amplitude. The end result is a file comprising a string of bytes, eg ...
1001 1110 0001 1010 0111 0100 1111 1101 etc

Numerizare sunet utilizând o rezoluție de eșantionare de 4 biți

Numerizarea sunetului

- **Opt biți** pot înregistra 256 de niveluri de amplitudine diferite. Acest lucru este adecvat pentru a surprinde un număr limitat de variații în amplitudine, cum ar fi cele dintre o șoaptă umană și un strigăt, dar sunt necesare rezoluții mai mari de eșantionare pentru a reproduce cu acuratețe sunete cu o gamă mai largă de amplitudini, cum ar fi spectacolele muzicale. Astfel, sunetul pe 8 biți este utilizat în general numai pentru sunete simple sau în aplicații multimedia care necesită dimensiuni foarte mici ale fișierelor.

- Standardul CD-Audio, care utilizează o rezoluție de eșantionare de 16 biți, acceptă peste 65 de mii de niveluri de amplitudine diferite, în timp ce standardul DVD-Audio pe 24 de biți poate reprezenta peste 16 milioane de niveluri.

Numerizarea sunetului

- Inadequate sample resolution can distort sound in two different ways: **quantization** and **clipping**.

- **Quantization** - each amplitude sample must be assigned one of the numbers available in the code being used. If the number of distinct values is too small, perceptually different amplitudes will be assigned the same number. Rounding a sample to the closest available value is known as **quantization**. In the case of sound, excessive quantization may produce a background hissing or a grainy sound. The solution is to record with a higher sample resolution (for instance, by using 16 rather than 8 bits).

Numerizarea sunetului

- **Clipping** - sound-sampling equipment is designed for a selected decibel range. If the source sound exceeds this range (as, for instance, when someone yells into a microphone held close to their lips), higher amplitudes cannot be encoded, because no values are available to represent them. The waveform of a clipped sound shows square tops and bottoms marking the point at which the highest amplitudes could not be captured. Clipping can produce a harsh, distorted sound.

Clipping occurs when wave amplitude exceeds available sample values.

Clipped waves have square tops and bottoms and a harsh sound.

Numerizarea sunetului

- The solution to **clipping** is to lower the amplitude of the source sound to record within the limits of the ADC circuitry.

- Recording equipment usually includes some form of meter such as a swinging needle, colored bars, or lights to show input levels and alert users when the amplitude range has been exceeded.

- The familiar, "Testing—one, two, three," is often used to establish the proper distance and speech level when recording with a microphone.

Numerizarea sunetului

- Clipping can also occur during the mixing of different audio tracks.

- **Mixing** is the process of combining two or more sound selections, or *tracks*, into a single track. For example, a background music track might be mixed with a voice track of a poetry reading. This combination of two or more tracks may produce an amplitude that exceeds the available range. Adjustments to the volume of each track can eliminate the problem.

- Another solution is to use higher sample resolutions (for instance, 24-bit) to provide a wider range of amplitude values.

Numerizarea sunetului

- **Frecvența de eșantionare (en: sample rate)** este reprezentată de numărul de măsurători realizate într-un interval de timp fix. O rată de un eșantion pe secundă este desemnată ca Hertz.
- Deoarece eșantioanele de sunet sunt preluate întotdeauna de mii de ori pe secundă, ratele de eșantionare sunt de obicei indicate în kilohertzi.
- Se determină pe baza teoremei lui Nyquist (minim dublul frecvenței maxime a sunetului)
- Frecvența de eșantionare afectează calitatea sunetului prin determinarea gamei de frecvențe care pot fi reprezentate într-o înregistrare digitală. Sunt necesare cel puțin două măsurători pentru a captura fiecare ciclu al unei unde sonore - una pentru fiecare valoare ridicată sau vârf și una pentru fiecare valoare scăzută sau jgheab. Cea mai mare frecvență care poate fi capturată este, astfel, jumătate din rata de eșantionare.

Numerizarea sunetului

- Sunetul de calitate CD captează 44.100 de eșantioane pe secundă (rata de eșantionare de 44,1 kHz) și poate reprezenta frecvențe de până la 22,050 Hz sau 22,05 kHz. DVD-Audio folosește o frecvență de eșantionare de 96 kHz pentru a capta frecvențe de până la 48 kHz.

- Sunetele care nu conțin frecvențe înalte pot fi reprezentate mai eficient folosind rate de eșantionare mai mici, deoarece acest lucru va produce o dimensiune totală mai mică a fișierului.

- O problemă potențială în cazul utilizării unei frecvențe de eșantionare mai reduse este aliasing-ul.

Numerizarea sunetului

Balancing File Size and Sound Quality

- It is not always necessary to use DVD-quality sound. For example normal speech, contain relatively low frequencies and a limited range of amplitudes. In this case, higher sample rates and sample resolutions do not improve sound quality, but they create needlessly large files. Using a rate of 11.025 kHz for voice recording results in a file only one-quarter the size of a CD-quality sound file. In addition, 8-bit sample resolution and monaural sound are usually adequate for speech. This further reduces file size by a factor of four, one-sixteenth the size of a stereo CD recording.

Resolution	Rate	Stereo/Mono	Size (1 Minute)	Quality
16	44.1 kHz	Stereo	10 MB	CD
16	22.05 kHz	Stereo	5 MB	FM radio
8	11.025 kHz	Mono	650 KB	AM radio
8	5.5 kHz	Mono	325 KB	Bad telephone

Compresia sunetului

- Reducerea **rezoluției de eșantionare** și a **frecvenței de eșantionare** sunt două moduri de a reduce dimensiunea unui fișier de sunet eșantionat. Aceste metode funcționează bine pentru sunete la frecvențe relativ joase și intervale de amplitudine înguste. Nu sunt eficiente pentru sunetele care conțin intervale mai largi de frecvență și amplitudine, cum ar fi spectacolele muzicale. În aceste cazuri poate fi utilizată o altă strategie: **compresia**.

- Compresia poate fi fie fără pierderi, fie cu pierderi. **Compresia fără pierderi** utilizează o codificare mai eficientă pentru a reduce dimensiunea unui fișier, păstrând în același timp toate informațiile din original. **Compresia cu pierderi** renunță la o parte din informațiile originale.

Compresia sunetului

- Deoarece algoritmii de **compresie cu pierderi** produc fișiere mult mai mici, acestea sunt tehnica preferată pentru compresia sunetului.

- Codecurile de compresie a sunetului cu pierderi folosesc diverse tehnici pentru a reduce dimensiunile fișierelor. Unele dintre acestea profită de psihoacustică, de interacțiunea dintre condițiile psihologice ale percepției umane și proprietățile sunetului. De exemplu, în timp ce oamenii cu auz optim pot percepe frecvențe de până la aproximativ 20 kHz, majoritatea oamenilor nu pot distinge frecvențe peste aproximativ 16 kHz. Aceasta înseamnă că informațiile cu frecvență mai mare pot fi eliminate fără ca majoritatea ascultătorilor să observe o diferență. Sunetele cu amplitudine mai mare într-un canal stereo vor „acoperi”, de obicei, sunete mai slabe prezente pe celălalt canal.

Compresia sunetului

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

2. 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 puternica datorită inerției timpanului.

Compresia sunetului

- Compresia cu pierderi folosește, de asemenea, alte tehnici, cum ar fi cuantificarea cu **rată de biți variabilă (VBR)**. În VBR, sunetele sunt codificate folosind un număr diferit de biți pe secundă, în funcție de complexitatea sunetului. Pentru pasaje simple de sunet cu frecvențe limitate, se folosește un număr mai mic de biți pe secundă decât pentru pasaje mai complexe, cum ar fi cele cu multe instrumente diferite și frecvențe mai mari.

- Algoritmii de codificare cu pierderi de informație, cum ar fi MP3, pot reduce dimensiunile fișierelor cu până la 80%, păstrând o calitate similară cu cea a unui CD audio.

Formate audio

- **MP3** (MPEG1, audio layer 3)
  - extenise: .mp3 or .mpga
  - format audio popular care oferă o compresie semnificativă păstrând în același timp o calitate excelentă.
  - utilizat frecvent în aplicații web.

Formate audio

- AAC (advanced audio coding)
  - extensie: .mp4 and .aac
  - succesorul MP3 specificat în standardul MPEG4, care oferă o calitate a sunetului mai bună decât MP3 la rate de biți comparabile.
  - de asemenea, poate reduce semnificativ dimensiunile fișierelor pentru o calitate audio comparabilă.
  - utilizat de: Apple iPod, Apple iPad, Apple iPhone, YouTube, and Sony PlayStation.

Formate audio

- Alte formate de fișiere audio:
  - WMA (Windows Media Audio);
  - WAVE - formatul standard de fişier audio ce conţine sunet în reprezentare PCM necomprimat;
  - AIFF (Audio Interchange File Format) - formatul standard pentru audio digital utilizat pe platformele Apple (variante: necomprimat / comprimat);
  - AU.

Optional

Synthesized Sound

Further reading: https://en.wikipedia.org/wiki/MIDI

Exemple de aplicații utilizate pentru editarea sunetului

- Adobe Audition:
  - software comercial;
  - https://www.adobe.com/products/audition.html

- Audacity
  - software open source;
  - https://www.audacityteam.org

Web Audio - <audio>

<audio> - elementul HTML este utilizat pentru a încorpora conținut sonor în aplicații web. Poate conține una sau mai multe surse audio, reprezentate folosind atributul "src" sau elementul <source>, dintre care browser-ul îl va alege pe cel mai potrivit.

```
<audio controls>
    <source src="media/viper.mp3" type="audio/mp3">
    <source src="media/viper.ogg" type="audio/ogg">
    <p>Your browser doesn't support HTML5 audio.</p>
</audio>
```

- încercați codul de mai sus în Internet Explorer, comentând formatul mp3

Web Audio

- Atributele elementului <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

- API: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/audio

Web Audio - HTMLAudioElement

HTMLAudioElement (JavaScript) - oferă acces la proprietățile elementelor <audio>, precum și la metodele aplicabile acestora. Este derivat din HTMLMediaElement.

- Proprietăți:
  - src / 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

Web Audio - HTMLAudioElement

- Proprietăți (continuare):
  - readyState – indică starea curentă a elementului
  - volume – permite citirea / modificarea volumului

- 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

Web Audio - HTMLAudioElement

- 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

- API: https://developer.mozilla.org/en-US/docs/Web/API/HTMLAudioElement

Lab Example:

- https://ase-multimedia.azurewebsites.net/audio-playlist/

Web Audio API

- **Web Audio API -** oferă facilități avansate pentru lucrul cu fișiere audio pe web, permițând dezvoltatorilor să aleagă surse audio, să adauge efecte audio, să creeze vizualizări ale sunetului.

- API: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API

Web Audio API

- 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 element <audio>, de la microfon 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>
  - **createMediaStreamSource(stream):** construiește un nod sursă pe baza unui stream
  - **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()**: închide audiocontext-ul, eliberând orice resurse audio de sistem utilizate.

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

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

Web Audio API

```
const context = new AudioContext();

navigator.mediaDevices.getUserMedia({ audio: true })
.then((stream) => {
    const microphone = context.createMediaStreamSource(stream);
    const filter = context.createBiquadFilter();
    // microphone -> filter -> destination
    microphone.connect(filter);
    filter.connect(context.destination);
});
```

Test the code on this slide in your browser

Web Audio API

- Creare de vizualizări: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Visualizations_with_Web_Audio_API
- Playground: http://webaudioplayground.appspot.com/
- Record audio: https://googlechromelabs.github.io/web-audio-samples/audio-recorder

Lab Example: Web Audio API

- https://ase-multimedia.azurewebsites.net/audio-web-audio-api/

WebRTC API

- WebRTC (Web Real-Time Communications) este o tehnologie care permite aplicațiilor și site-urilor web să capteze și să transmită conținut audio și / sau video.

- Setul de standarde care cuprinde WebRTC face posibilă efectuarea de teleconferințe peer-to-peer, fără a fi necesar ca utilizatorul să instaleze plugin-uri sau orice alt software terță parte.

- Web RTC API: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API, https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia

Demo

- https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Taking_still_photos

Lab Example: Speech API

- https://ase-multimedia.azurewebsites.net/speech-api/

Alte exemple

- https://googlechromelabs.github.io/web-audio-samples/

Recomandări

- În general nu este recomandată utilizarea atributului “autoplay”

- Este necesar ca utilizatorii:
  - Să poată controla volumul sunetului.
  - Să poată opri / porni redarea sunetului.

- Se va evita utilizarea excesivă a sunetului. Sunetul poate fi mai obositor pentru utilizatori decât imaginile sau textul.

Video digital

Definiție

- Cuprinde totalitatea tehnicilor de captură, procesare și stocare a imaginilor în mișcare (precum și a sunetului asociat) prin intermediul unui dispozitiv de calcul.
- Video digital cuprinde o serie de imagini digitale afișate în succesiune rapidă. În contrast, filmul analogic, folosește o serie de fotografii afișate în succesiune rapidă.
- Video digital a fost introdus pentru prima dată comercial în 1986 cu formatul Sony D1.

Avantaje

- 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 (interlaced sau progresiv)
- Metode de compresie

Caracteristici - Rezoluția

- **lățimea** și **înălțimea** imaginii afișate, măsurate în pixeli. Cu alte cuvinte, numărul total de pixeli conținuți în fiecare cadru individual (numit și **rezoluție spațială**)[1]

- cu cât rezoluția este mai redusă, cu atât puterea de procesare necesară și spațiul de stocare sunt mai reduse [2].

- Rezoluția formatului DV (video digital) este de **720 × 480 pixeli**. Este astfel necesar ca dispozitivul de redare să proceseze și să transmită informații pentru aproape **350,000 pixeli** la rate de până la 30 de ori pe secundă[2].

Opțional

Characteristics – Common Screen Resolution

Name	Pixels (width x height)	Aspect Ratio	Notes
Standard Definition (SD)			
480p / 480i	720×480 (or 704×480)	4:3 (approx)	NTSC
576p / 576i	720×576 (or 704×576)	4:3 (approx)	PAL
High Definition (HDTV)			
720p	1280×720	16:9	
1080p / 1080i	1920×1080	16:9	
Ultra High Definition (UHDTV)			
4K (2160p)	3840×2160	16:9	Exactly 4 × 1080p
8K (4320p)	7680×4320	16:9	Exactly 16 × 1080p
8640p	15360×8640	16:9	Exactly 32 × 1080p
Digital Cinema (DCI)			
2K	2048 × 1080	1.90:1	The first generation of digital cinema projectors.
4K	4096 × 2160	1.90:1	2nd generation digital cinema.

Caracteristici – Rezoluție

Caracteristici – Rezoluție

Caracteristici - Spațiul de culoare și numărul de biți per pixel

- Spațiul de culoare este numărul de biți utilizați pentru a indica culoarea unui singur pixel sau numărul de biți utilizați pentru fiecare componentă de culoare a unui singur pixel

- Similar cu imaginile raster, cu cât sunt utilizați mai mulți biți pentru stocarea culorii, cu atât pot fi reproduse variații mai subtile ale culorilor.

Caracteristici – Numărul de cadre pe secundă

- Reprezintă frecvența la care un dispozitiv afișează imagini consecutive numite cadre.

- Percepția mișcării continue în videoclip este dependentă de doi factori. În primul rând, imaginile trebuie afișate suficient de rapid pentru a permite persistenței vederii să umple intervalul dintre cadre. În al doilea rând, schimbările în locația obiectelor de la cadru la cadru trebuie să fie graduale.

Caracteristici – Frame Rate

- O frecvență obișnuită pentru fișierele video este de 30 de cadre pe secundă (fps). Videoclipurile destinate streamingului pe Internet sunt adesea livrate la o rată de doar 15 fps, reducând efectiv volumul de date necesar la jumătate[1]

- NTSC utiliza aproximativ 30 de cadre pe secundă, în timp ce PAL utiliza 25 de cadre pe secundă [2].

Characteristics – Interlaced / Progressive

- Progressive video
  - each frame is displayed similar to the text on a page - line by line, top to bottom.

- Interlaced video
  - each frame is composed of two halves of an image. In the first pass all odd numbered lines are displayed, from the top left corner to the bottom right corner. The second pass displays the second and all even numbered lines, filling in the gaps in the first scan.

  - the two halves are referred to individually as fields. Two consecutive fields compose a full frame. If an interlaced video has a frame rate of 15 frames per second the field rate is 30 fields per second.

Opţional

Characteristics – Interlaced / Progressive

Characteristics – Interlaced / Progressive

- Examples of formats:
  - PAL -576p (progressive) / 576i (interlaced)
  - FullHD – 1080p (progressive) / 1080i (interlaced)

- Benefits of interlacing
  - for a fixed bandwidth, interlace provides a video signal with twice the display refresh rate for a given line count (versus progressive scan video at a similar frame rate—for instance 1080i at 60 half-frames per second, vs. 1080p at 30 full frames per second).
  - bandwidth benefits **only apply** to an **analog** or **uncompressed digital video** signal. With digital video compression, as used in all current digital TV standards, interlacing introduces additional inefficiencies.

Characteristics – Interlaced / Progressive

- Interlacing issues
  - Because each interlaced video frame is composed of two fields captured at different moments in time, interlaced video frames can exhibit motion artifacts known as interlacing effects, if recorded objects move fast enough to be in different positions when each individual field is captured.
  - These artifacts may be more visible when interlaced video is displayed at a slower speed than it was captured, or in still frames.

Caracteristici – Compresia

- Scopul compresiei video:
  1. reducerea dimensiunii fișierelor video prin eliminarea informației redundante.
  2. păstrarea calității percepute a imaginii.

Caracteristici – Compresia

- **Redundanța** poate apărea:
  - în fiecare cadru considerat individual (similar imaginilor) – *redundanță spațială*.
  - de-a lungul timpului într-o secvență video care conține multe cadre – *redundanță temporală*.

- **Examplu:** o înregistrare de cinci secunde (150 de cadre) a unui cer albastru într-o zi fără nori.
  - mii de pixeli albaștri păstrează aceeași valoare a culorii pe întreaga secvență de 150 de cadre. Acest fenomen se numește **redundanță temporală** și apare ori de câte ori valoarea unui pixel rămâne neschimbată de la un cadru la altul într-o secvență bazată pe timp. De asemenea, pixelii mulți pixeli au aceași culoare în interiorul fiecărui cadru. Acest fenomen se numește **redundanță spațială**

Caracteristici – Compresia

Redundanța spațială (intra-cadru)

Caracteristici – Compresia

Redundanță temporală (1 second / 30 Frames)

Caracteristici – Compresia

- *Compresie Intraframe* (or *I-frame*)
  - Elimină redundanța spațială „în interiorul” unui cadru video în același mod în care compresia JPEG este utilizată pentru a reduce redundanța într-o fotografie digitală.
  - I-frames sunt de obicei comprimate la un raport de 10:1. Aceasta înseamnă că un cadru I-frame comprimat consumă doar 10% din spațiul necesar unui cadru necomprimat.
  - Deoarece cadrele I-frame sunt complet definite de informațiile stocate, ele pot fi ușor decodate și redate pe ecran.

Caracteristici – Compresia

- *Compresie Interframe* (metoda cel mai frecvent utilizată)
  - exploatează atât redundanțele **spațiale**, cât și cele **temporale**
  - folosind metoda anterioară de compresie intraframe, toate cadrele dintr-un flux video sunt transformate în cadre de tip I-frame. Fiecare cadru este comprimat prin eliminarea redundanței spațiale. Astfel, compresia se aplică uniform tuturor cadrelor dintr-un flux video.
  - cu **compresie interframe**, cadrele I-frame sunt create la intervale fixe (de obicei la fiecare 15 cadre). Un cadru I-frame marchează începutul unei secvențe împachetate de cadre adiacente numite GOP (Group of Pictures). Cadrul I-frame complet definit servește drept cadru cheie sau referință pentru celelalte cadre din grup. Sarcina sa este să mențină valorile pixelilor care nu se vor schimba pe parcursul secvenței. Culoarea pixelilor se va modifica doar dacă acestă modificare este stocată explicit în cadrele următoare.

Caracteristici – Compresia

- Compresia MPEG
  - exploatează atât redundanțele spațiale, cât și cele temporale (compresie interframe)
  - fiecare cadru **I-frame** este urmat de o secvență de 14 cadre desemnate drept cadre **B-frame** sau **P-frame**.

Caracteristici – Compresia

- *P-frame*
  - o imagine codificată predictivă care stochează doar date pentru pixeli diferiți față de cadrul precedent (I-frame).
  - Examplu: într-o înregistrare a unei păsări care zboară pe un cer albastru, numai pixelii afectați de mișcarea păsării vor fi înregistrați în P-frame.

Caracteristici – Compresia

- *B-frame*
  - o imagine codificată predictivă bidirecțională. Înregistrează modificările atât față de cadrul precedent, cât și de cel imediat următor.
  - În medie, un cadru B-frame poate fi comprimat de două ori mai mult decât un cadru P-frame.

- În cazul **compresiei interframe**, cadrele nu sunt comprimate la fel de mult. Cadrele P-frame și cadrele B-frame sunt mai puternic comprimate și necesită mai multă putere de procesare pentru codificare și decodare.

Caracteristici – Compresia

Caracteristici – Compresia

Display order	I	B	B	P	B	B	P	B	B	I
Coding and transmission order	I	P	B	B	P	B	B	I	B	B

Compresia video

Why Snow and Confetti Ruin YouTube Video Quality (video recomandat de unul dintre colegii voștri)

Caracteristici – Formate Video- Containere

- Container:
  - specifică structura de stocare a componentelor video (imagine + audio) și a datelor asociate (metadate , subtitrări,..)

Caracteristici – Formate Video- Containere

- Matroska Multimedia Container MKV:
  - extensii: .mkv .mk3d .mka .mks
  - formatul *.webm propus de Google are la bază acest format
  - format deschis
  - acceptă aproape orice format audio sau video, ceea ce îl face adaptabil, eficient și considerat unul dintre cele mai bune moduri de stocare a fișierelor audio și video.
  - acceptă mai multe fișiere audio, video și subtitrare, chiar dacă acestea sunt codificate în formate diferite.
  - datorită opțiunilor oferite, precum și gestionării erorilor (posibilitate de redare a fișierelor corupte), a devenit rapid unul dintre cele mai bune containere disponibile în prezent.

Caracteristici – Formate Video- Containere

- MPEG-4 (MP4):
  - extensie: .mp4
  - dezvoltat de către Motion Pictures Expert Group și utilizat inițial de QuickTime
  - cel mai frecvent utilizat tip de container pentru a stoca video și audio
  - utilizează codificarea MPEG-4 sau H.264, precum și AAC sau AC3 pentru audio.
  - este acceptat pe scară largă pe majoritatea dispozitivelor de consum și cel mai comun container utilizat pentru videoclipuri online

Caracteristici – Formate Video- Containere

- Alte containere video populare:
  - Ogg - format container gratuit, deschis
  - AVI – Windows
  - MOV – Mac everything
  - MPEG or MPG – creat de MPEG group
  - FLV – Flash video
  - VOB – DVDs

- Comparație:
  - https://en.wikipedia.org/wiki/Comparison_of_video_container_formats

Web Video

*"**Flash** was created during the PC era – for PCs and mice. Flash is a successful business for Adobe, and we can understand why they want to push it beyond PCs. But the mobile era is about **low power devices**, **touch interfaces** and **open web standards** – all areas where Flash falls short. The avalanche of media outlets offering their content for Apple's mobile devices demonstrates that Flash is no longer necessary to watch video or consume any kind of web content."*

Steve Jobs

Web Video - The Evolution Of Flash

Web Video

- <video> - Elementul HTML utilizat pentru a încorpora conținut video într-un document. Elementul video conține una sau mai multe surse video. Pentru a specifica o sursă video, se va utiliza fie atributul **src**, fie elemente de tip **<source>** dintre care browser-ul va alege formatul cel mai potrivit.

```
<video controls>
  <source src="foo.webm" type="video/webm">
  <source src="foo.ogg" type="video/ogg">
  <source src="foo.mov" type="video/quicktime">
  I'm sorry; your browser doesn't support HTML5 video.
</video>
```

Web Video

- <video>
  - atribute: *autoplay, controls, src, volume, ...*
  - API: *https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video*

- HTMLVideoElement (JavaScript)
  - poperties: currentSrc, currentTime, duration, ended, error, paused, readyState, volume
  - methods canPlayType, load, pause, play
  - events: canplay, ended, pause, play, volumechange, waiting

  - API: https://developer.mozilla.org/en-US/docs/Web/API/HTMLVideoElement

Web Video

- <track>
  - HTML element used as a child of the media elements - <audio> and **<video>**. It lets you specify timed text tracks (or time-based data), for example to automatically handle subtitles.
  - The tracks are formatted in WebVTT format (.vtt files) - Web Video Text Tracks.
  - Link: *https://developer.mozilla.org/en-US/docs/Web/HTML/Element/track*

Web Video

- <source>
  - HTML element is used to specify multiple media resources for either the <picture>, the <audio> or the **<video>** element.
  - It is an empty element.
  - It is commonly used to serve the same media content in multiple formats supported by different browsers..
  - Link: *https://developer.mozilla.org/en-US/docs/Web/HTML/Element/source*

```
<video controls>
  <source src="foo.webm" type="video/webm">
  <source src="foo.ogg" type="video/ogg">
I'm sorry; your browser doesn't support HTML5 video.
</video>
```

Web Video

- recommended video formats:
  - webm – type = video/webm
  - mp4 – type = video/mp4
  - .ogg – type = video/ogg

```
<video controls>
  <source src="foo.webm" type="video/webm">
  <source src="foo.ogg" type="video/ogg">
I'm sorry; your browser doesn't support HTML5 video.
</video>
```

Web Video

- Live Demo: https://www.w3.org/2010/05/video/mediaevents.html
- API: https://www.w3.org/TR/html5/embedded-content-0.html#mediaevents
- Media Player:
  - https://developer.mozilla.org/en-US/Apps/Fundamentals/Audio_and_video_delivery/Adding_captions_and_subtitles_to_HTML5_video

Web Video

```
var W = canvas.width = video.videoWidth;
var H = canvas.height = video.videoHeight;

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

for (let y = 0; y < H; y++) {
    for (let x = 0; x < W; x++) {
        const i = (y * W * 4) + x * 4;
            // change values in imageData.data[i+...]
        }
    }
}
context.putImageData(imageData, 0, 0);
// other canvas drawing operations
```

Web Video

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

// change the video file
v[0].src = "media/test.mp4";
v[0].load();
v[0].play();
```

Web Video – Simple playlist

```
var lista = ["movie.mp4", "v2.mp4"];
var index = 0;

var video = document.querySelector("#video");

video.addEventListener("ended", function () {
      index = index + 1;
      if (index >= lista.length) index = 0;

      video.src = lista[index];
      video.load();
      video.play();
});
```

Lab Examples

- https://ase-multimedia.azurewebsites.net/video-player/
- https://ase-multimedia.azurewebsites.net/video-processing/
- https://ase-multimedia.azurewebsites.net/video-effects/

Animația

Animația

- Animation is the process of creating the illusion of motion and the illusion of change

- Main techniques:
  - movie technique
  - key frames
  - color changing

Digital Animation

- Digital animation takes two different forms:
  - two-dimensional - has evolved from traditional techniques, particularly cel animation;
  - three-dimensional - exploits the capabilities unique to the computer to produce an entirely new form of animation.

Digital Animation

- 1. frame-by-frame animation
  - animators produce each successive frame manually
  - technique that provides complete control over frame content, but is also very time consuming

Digital Animation

- 2. tween animation
  - the animator creates the key frames and the computer automatically produces the tweens.
  - There are several different types of tweens:
    - motion tween - can be used to move an object from one position to another. The first key frame places the object in one position and the second places it in another. The program then fills in the intervening frames.
    - path-based animation: the animator draws a path from one key frame to another and the computer fills in the intervening images, spaced out along the path, to produce the desired motion
    - morphing - the shape of one image is gradually modified until it changes into another shape

Digital Animation

- size tweening - the first key frame is the object at its initial size and the second is the final size.

- alpha tweening- color and transparency can be animated using key frames representing initial and final image properties.

- 3. programmed animation

Web Animation

Called at a particular time interval using one of the functions:

- *setInterval(function, interval_ms)*
- *setTimeout(function, interval_ms)*
- *requestAnimationFrame(function)*

Demo

- lab animation examples

CSS Transitions

- Guide: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Transitions/Using_CSS_transitions

CSS Animations

- API: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations
- Guide: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations/Using_CSS_animations

Web Animations API

- The **Web Animations API** allows for synchronizing and timing changes to the presentation of a Web page, i.e. animation of DOM elements.

- Limited browser support: https://caniuse.com/#feat=web-animation (compared to **CSS Animations** with https://caniuse.com/#feat=css-animation)

- API: https://developer.mozilla.org/en-US/docs/Web/API/Web_Animations_API

Demo

https://css-tricks.com/css-animations-vs-web-animations-api/

Progressive Web Apps

Progressive Web Apps

- An application developed using standard web technologies (HTML5, JavaScript, CSS3) which is:
  - **Reliable** - load instantly even in uncertain network conditions.
  - **Fast** - respond quickly to user interactions with silky smooth animations and no janky scrolling.
  - **Engaging** - feel like a natural app on the device, with an immersive user experience.

Reliable

- Progressive web apps are not dependent on the user's connection like traditional websites are.

- When a user visits a progressive web app, it will register a **service worker** that can detect and react to changes in the user's connection. It can provide a fully featured user experience for users who are offline, online, or suffering from an unreliable connection.

- A **service worker**, written in JavaScript, is like a **client-side proxy** and puts you in control of the cache and how to respond to resource requests. By pre-caching key resources you can eliminate the dependence on the network, ensuring an instant and reliable experience for your users.

Reliable

- When launched from the user's home screen, **service workers** enable a Progressive Web App to load instantly, regardless of the network state.

Reliable

- A service worker, written in JavaScript, is like a client-side proxy and puts you in control of the cache and how to respond to resource requests. By pre-caching key resources you can eliminate the dependence on the network, ensuring an instant and reliable experience for your users.

Fast

- Using service workers, progressive web apps can launch in an instant, whether the user has a blazing fast connection, an unreliable 2G connection, or even no connection at all. Sites can load in milliseconds, much faster than anything we have experienced on the web before, and sometimes faster than native apps.

Engaging

- Push notifications
  - Progressive web apps can send notifications to their users. The notifications have a completely native feel and are indistinguishable from native app notifications.

- Homescreen shortcut
  - Once a user has shown interest in a progressive web app, the browser will automatically suggest that he add a shortcut to his homescreen—completely indistinguishable from any native app.

Engaging

Desktop - **Homescreen shortcut**

Engaging

Mobile - **Homescreen shortcut**

Engaging

- Native look
  - Progressive web apps launched from the homescreen can have an entirely native, app-like look. They can have a splash screen as they are loading. They can launch in full-screen mode, without the browser and phone UI around them. They can even lock themselves to a specific screen orientation (a vital requirement for games).

Service Worker

- At the heart of every progressive web app is the service worker.
- Before service workers, we had code running either on the server or in the browser window. Service workers introduce another layer.
- A service worker is a script that can be registered to control one or more pages of your site. Once installed, a service worker sits outside of any single browser window or tab.
- From this place, a service worker can listen and act on events from all pages under its control. Events such as requests for files from the web can be intercepted, modified, passed on, and returned to the page

Service Worker

Pages communicating with a service worker while the user is **offline**

Checking the tabs associated with a service worker

- Using the Developer Tools in Google Chrome we can easily check the browser tabs corresponding to a service worker

Service Worker

The service worker communicating with a server after a user has left the page

Simulating an offline state in Google Chrome

Service Worker

- Documentation:
  - https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API

- A service worker is run in a worker context: it therefore has no DOM access, and runs on a different thread to the main JavaScript that powers your app, so it is not blocking.

- It is designed to be fully async; as a consequence, APIs such as synchronous XMLHttpRequest and Web Storage API can't be used inside a service worker.

- Service workers only run over HTTPS, for security reasons.

Service Worker

- Registration:
  - A service worker is first registered using the ServiceWorkerContainer.**register**() method. If successful, the service worker will be downloaded to the client and attempt installation/activation for URLs accessed by the user inside the whole origin, or inside a subset specified by you.

  - Documentation:
    - https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorkerContainer/register

Service Worker

- Note: check the documentation for an example also declaring a scope

```
if ("serviceWorker" in navigator) {
  navigator.serviceWorker.register("/serviceworker.js")
    .then(function(registration) {
      console.log("Service Worker registered with scope:", registration.scope);
    }).catch(function(err) {
      console.log("Service worker registration failed:", err);
    });
}
```

Demo

- Create an empty html page and try to register a service worker
  - Note: the registration will fail because we have not yet created the "serviceworker.js" file

```
Service worker registration failed: TypeError: Failed to register a          index.html:20
ServiceWorker: A bad HTTP response code (404) was received when fetching the script.
```

Service Worker

- Verify that the current browser supports service workers

```
if ("serviceWorker" in navigator) {
```

- The register call returns a *promise*. If the promise is fulfilled, meaning the service worker was registered successfully, the function defined in the then statement is called. If there was any problem, the function defined inside the catch block would be executed.

```
navigator.serviceWorker.register("/serviceworker.js")
```

- If there was any problem, the function defined inside the catch block would be executed.

Service Worker

- Intercept requests from the Service Worker

```
self.addEventListener("fetch", function(event) {
    console.log("Fetch request for:",
event.request.url);
});
```

- The code adds an event listener to our service worker by calling addEventListener on self (self within a service worker refers to the service worker itself).

- The function we define to handle these events accesses the request object (available as a property of the fetch event) and logs the URL of that request.

Demo

- Create a service worker and subscribe to the “fetch” event

Service Worker

- every request made by our page (including to third-party servers) now passes through our service worker. All of those requests can now be intercepted, analyzed, and even manipulated. ☺

```
self.addEventListener("fetch", function(event) {
  if (event.request.url.includes("bootstrap.min.css")) {
    event.respondWith(
      new Response(
        "div {background: green;}",
        { headers: { "Content-Type": "text/css" }}
      )
    );
  }
});
```

Service Worker

- Respond to requests with content from the web:

```
self.addEventListener("fetch", function(event) {
  if (event.request.url.includes("/img/logo.png")) {
    event.respondWith(
      fetch("/img/logo-rotated.png")
    );
  }
});
```

Fetch API

- Documentation: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
- Provides an interface for fetching resources (including across the network).
- Similar to **XMLHttpRequest**, but the new API provides a more powerful and flexible feature set.
- Moreover, **XMLHttpRequest** is **not** available in ServiceWorkers.

Fetch API

```
// Fetch by URL
fetch("/img/logo.png");
// Fetch by the URL in the request object
fetch(event.request.url);
// Fetch by passing a request object.
// In addition to the URL, the request object may contain additional
headers,
// form data, etc.
fetch(event.request);
```

- The first argument in fetch is mandatory and can contain either a request object, or a string with a relative or absolute URL

Fetch API

- The second argument is **optional** and can contain an object with options for the request.

```
return fetch(url, {// Default options are marked with *
        method: "POST", // *GET, POST, PUT, DELETE, etc.
        mode: "cors", // no-cors, cors, *same-origin
        cache: "no-cache", // *default, no-cache, reload, force-cache, only-if-cached
        credentials: "same-origin", // include, *same-origin, omit
        headers: {
                "Content-Type": "application/json; charset=utf-8",
                // "Content-Type": "application/x-www-form-urlencoded",
        },
        redirect: "follow", // manual, *follow, error
        referrer: "no-referrer", // no-referrer, *client
        body: JSON.stringify(data), // body data type must match "Content-Type" header
})
.then(response => response.json()); // parses response to JSON
```

Service Worker - Capturing Offline Requests

- Handle the communication error using “catch”

- Simple “text” response:

```
self.addEventListener("fetch", function(event) {
  event.respondWith(
    fetch(event.request).catch(function() {
      return new Response(
      "There seems to be a problem with your connection.\n"
      );
    })
  );
});
```

Service Worker - Capturing Offline Requests

```
var responseContent =
  "<html><body>" +
  "<style>body {text-align: center; background-color: #333; color: #eee;}" +
  "</style>" +
  "<h1>Progressive Web Apps</h1>" +
  "<p>There seems to be a problem with your connection.</p>" +
  "</body></html>";

self.addEventListener("fetch", function(event) {
  event.respondWith(
    fetch(event.request).catch(function() {
      return new Response(
        responseContent,
        {headers: {"Content-Type": "text/html"}}
      );
    })
  );
});
```

Demo

- Check what happens if we remove the headers

- Note:
  - Most web servers are configured to automatically serve most common file types with the correct headers automatically.
  - When a server sends an HTML file, it constructs a response that contains both the HTML, as well as many headers, including a Content-Type header letting the browser know what to do with the response.
  - Because we are constructing a response from scratch, it is up to us to set the correct headers.

Service Worker

- If we modify the Service Worker, the changes **don't immediately** take effect after refreshing the browser. This is because the **old service worker** is still the *active* one, while your new service worker remains in a *waiting* state until the old one is no longer controlling the page.

- To ease development, we can tell the browser to let new service workers take immediate control of the page. In Chrome, this can be done by opening the Application tab of the developer tools, and under the Service Workers section enabling "Update on reload". This makes sure that each time we change the service worker and refresh the page, the new service worker will immediately take control of the page.

Service Worker

Turning on **Update on reload**

Service Worker LifeCycle

- The lifecycle of a service worker after being downloaded:

Service Worker LifeCycle

- Download
  - the service worker is immediately downloaded when a user first accesses a service worker–controlled site/page.

Service Worker LifeCycle

- Installing state
  - When a new service worker is registered using navigator.serviceWorker.register, the JavaScript is downloaded, parsed, and enters the **installing state**.
  - If the installation succeeds, the service worker will proceed to the **installed state**.
  - The **state** can be extended by calling **event.waitUntil()** and passing it a promise.

```
self.addEventListener('install', function(e) {
e.waitUntil(
    caches.open(cacheName).then(function(cache) {
       return cache.addAll(filesToCache);
    })
  );
});
```

Service Worker LifeCycle

- If an error occurs during installation, the script will instead be moved to the redundant state.

```
self.addEventListener("install", function() {
        console.log("install");
        throw 'error';
});
```

Service Worker LifeCycle

- Installed/waiting state
  - Once a service worker has successfully installed, it enters the **installed state**.
  - It will then immediately move on to the **activating state** unless another active service worker is currently controlling this app, in which case it will remain **waiting**.

Service Worker LifeCycle

- It is only move to the **activating state** when there are no longer any pages loaded that are still using the old service worker.
  - **Why?** Imagine a scenario in which you release a new version of a service worker, and this service worker's install event deletes userdata.json from the cache, adds users.json instead, and changes the fetch event to return the new file when user data is requested. If multiple service workers controlled different pages, the ones controlled by the old service worker might look for the old userdata.json file in the cache after it was removed, causing your app to break.

- Activation can happen sooner using **ServiceWorkerGlobalScope.skipWaiting()** and existing pages can be claimed by the active worker using **Clients.claim()**.

Service Worker LifeCycle

- Activating state
  - Before a service worker becomes active and takes control of the app, the **activate event** is triggered.
  - The **activating state** can be extended by calling **event.waitUntil()** and passing it a promise.

Service Worker LifeCycle

- Activating state
  - Before a service worker becomes active and takes control of the app, the **activate event** is triggered.
  - The **activating state** can be extended by calling **event.waitUntil()** and passing it a promise.

```
self.addEventListener('activate', function(e) {
e.waitUntil(
    caches.keys().then(function(keyList) {
      return Promise.all(keyList.map(function(key) {
        if (key !== cacheName) {
          return caches.delete(key);
        }
      }));
    })
  );
  return self.clients.claim();
});
```

Service Worker LifeCycle

- Activated state
  - Once a service worker is **activated**, it is ready to take control of the page and listen to functional events (such as **fetch**).
  - Note: a service worker can only take control of pages before they start loading. This means that pages that began loading before the service worker became active cannot be controlled by it.

Service Worker LifeCycle

- Redundant state
  - Service workers that failed during registration, or installation, or were replaced by newer versions, are placed in the **redundant state**. Service workers in this state no longer have any effect on your app.

Service Worker

- Further reading:
  - Service Worker Cookbook (Mozilla) - collection of working, practical examples of using service workers in modern web sites: https://serviceworke.rs/

CacheStorage API

- Documentation:
  - https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/cache-api

- The **Cache API** was created to enable Service Workers to cache network requests so that they can provide appropriate responses even while offline. However, the API can also be used as a general storage mechanism.

CacheStorage API

- How to check whether the API is available?

```
const cacheAvailable = 'caches' in self;
```

- Creating and opening a cache

```
caches.open('my-cache').then((cache) => {
  // do something with cache...
});
```

CacheStorage API

- Retrieving from a cache using **cache.match**

```
cache.match(request).then((response) => console.log(request, response));
```

- Retrieving all matching responses using **cache.matchAll**.

```
cache.matchAll(request).then((responses) => {
  console.log(`There are ${responses.length} matching responses.`);
});
```

CacheStorage API

- Adding to a cache
  - There are three ways to add an item to a cache - **put**, **add** and **addAll**. All three methods return a Promise.
  - cache.put

```
cache.put('/test.json', new Response('{"foo": "bar"}'));
```

  - cache.add
  - cache.addAll

CacheStorage API

```
let cacheName = 'bricks-final-1';
let filesToCache = [
  'index.html',
  'app.js',
  'styles.css',
  'img/icons/icon-152x152.png',
  'img/icons/icon-192x192.png'
];
```

Path	Content-Type	Content-Length	Time Cached
/canvas-game/app.js	application/x-javascript	2,098	11/25/2018, 12:22:31 AM
/canvas-game/img/icons/icon-152x152.png	image/png	5,103	11/24/2018, 11:56:25 PM
/canvas-game/img/icons/icon-192x192.png	image/png	6,786	11/25/2018, 12:22:31 AM
/canvas-game/index.html	text/html	696	11/25/2018, 12:22:31 AM
/canvas-game/lib/jquery-3.2.1.min.js	application/x-javascript	38,753	11/24/2018, 11:56:25 PM
/canvas-game/styles.css	text/css	232	11/25/2018, 12:22:31 AM

Checking the **Cache Storage** in Google Chrome

CacheStorage API

- Deleting an item

```
cache.delete(request);
```

  - Where request can be a Request or a URL string.

CacheStorage API

- Deleting a cache

```
caches.delete(key)
```

  - The function returns a Promise that resolves to true if the cache existed and was deleted, or false otherwise.

CacheStorage API

```
//2. Delete the old version of the cache
self.addEventListener('activate', function(e) {
e.waitUntil(
    caches.keys().then(function(keyList) {
      return Promise.all(keyList.map(function(key) {
        if (key !== cacheName) {
            return caches.delete(key);
        }
      }));
    })
  );
  return self.clients.claim();
});
```

Removing old caches

Lighthouse

- an open-source, automated tool for improving the quality of Progressive Web Apps, that eliminates much of the manual testing that was previously required.

- integrated in Google Chrome.

Is your App a Progressive Web App?

- Checklist: https://developers.google.com/web/progressive-web-apps/checklist
- Most of the checks are already included in Lighthouse ☺

Examples

Social Media – Twitter

- The Twitter Lite application is a PWA: https://blog.twitter.com/engineering/en_us/topics/open-source/2017/how-we-built-twitter-lite
- Windows Store Application
  - https://www.microsoft.com/ro-ro/p/twitter/9wzdncrfj140
- Twitter Lite for Android
  - https://play.google.com/store/apps/details?id=com.twitter.android.lite&hl=en&gl=US

Examples

Drawing- Google Chrome Canvas

- https://canvas.apps.chrome/

Further reading

Electron

https://www.electronjs.org/docs/latest/tutorial/tutorial-first-app

Web Storage API

Web Storage API

- Documentation:
  - https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API