



**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>

# Recommended Reading / Watching

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

Aspecte administrative

## Further Reading / Watching

- Cursuri Microsoft Virtual Academy - [mva.microsoft.com](http://mva.microsoft.com)
  - Free
- Cursuri PluralSight - [www.pluralsight.com](http://www.pluralsight.com)
  - Free trial
  - Free access (limited period) through [Microsoft DreamSpark](http://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 Digitale, 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

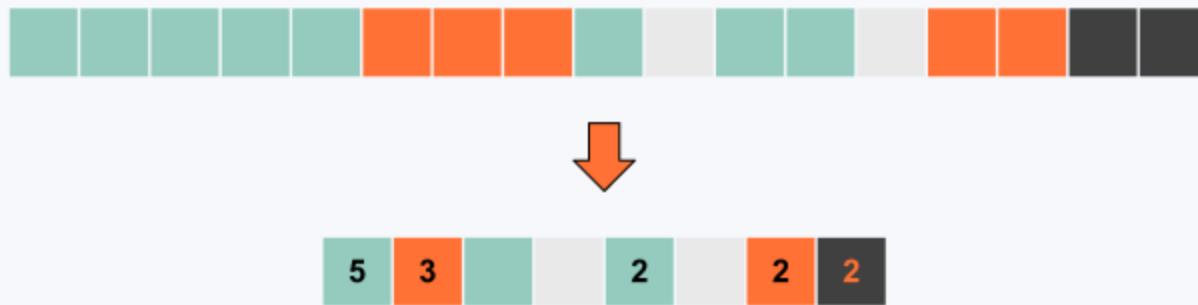
# Compresia

- 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 decompresionat, 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). Imaginele 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, aşa-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 perceptiei 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 intermediu 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# 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, comunicări;
  - 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 convertită și salvată sub formă digitală.
  - tipuri: flatbed / rotativ
  - operație: 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 semnauul analogic. Semnalul analogic este apoi convertit într-un semnal digital de către un alt dispozitiv numit **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

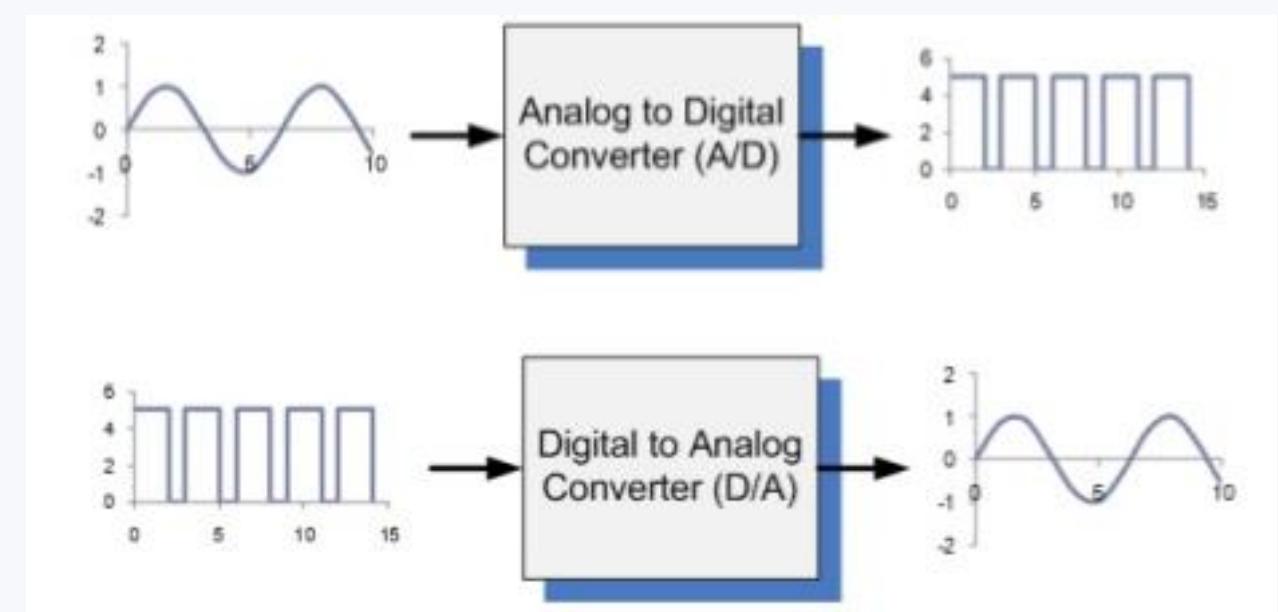
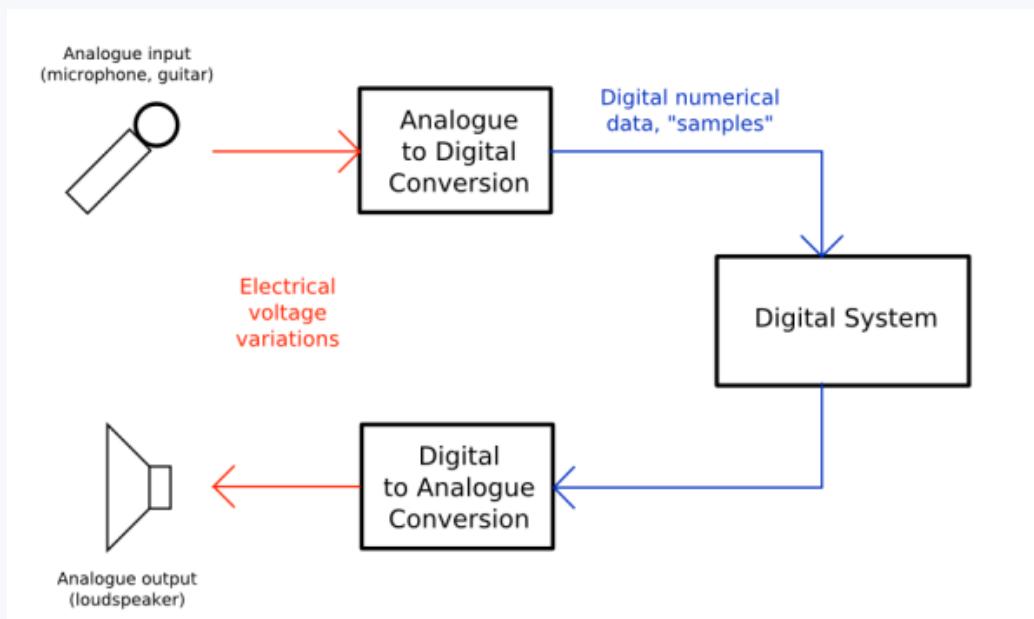


## Dispozitive periferice - Imagine

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



## Dispozitive periferice - Video

- Placă video
- Cameră video digitală

# Precondiții software

- **Drive** - 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, permitând 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:**
  - biblioteci 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

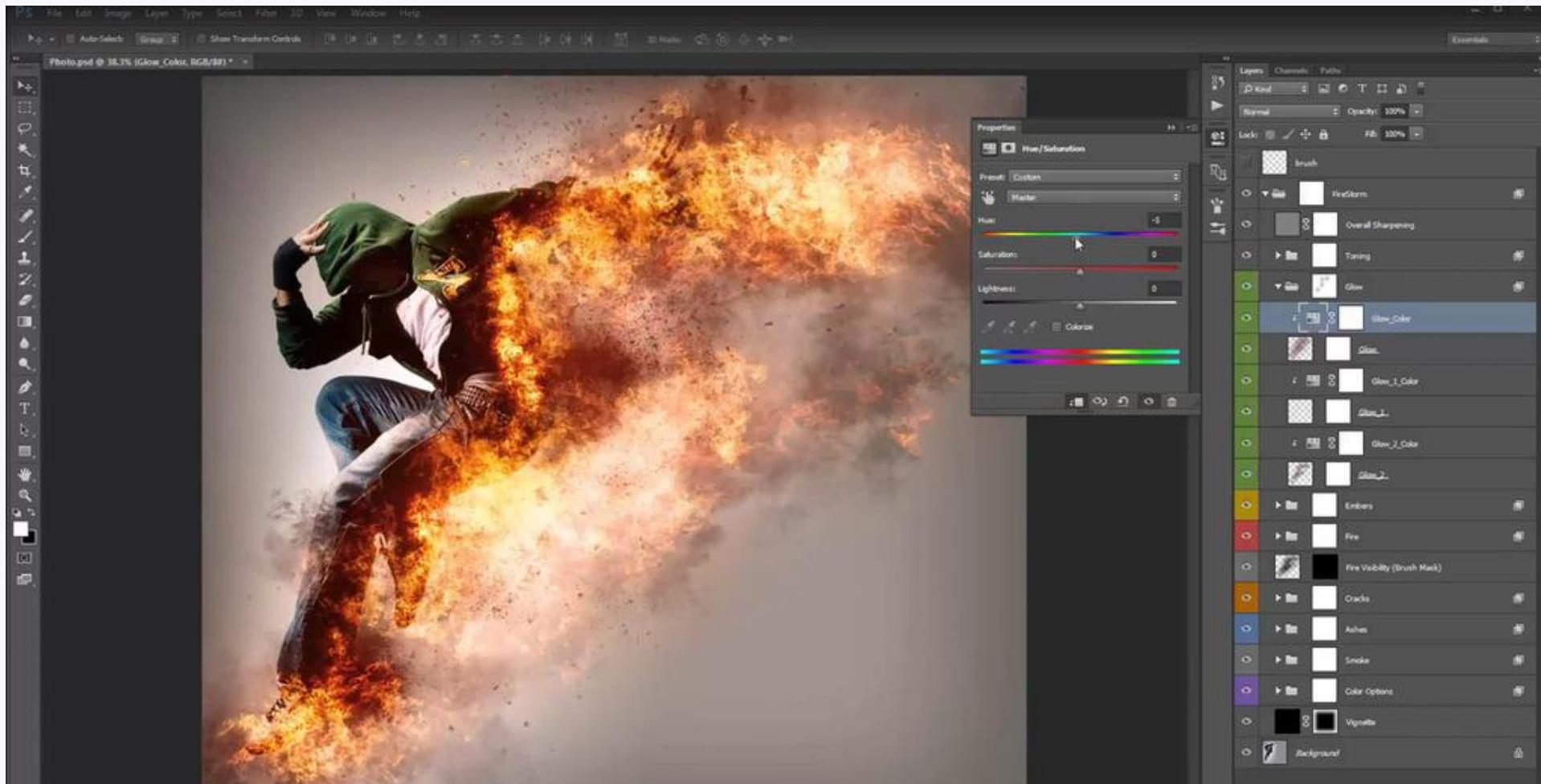
- Achiziția și prelucrarea de imagini 2D sau 3D.
- Categorii:
  - prelucrare imagini raster;
  - prelucrare imagini vectoriale;
  - lucru cu imagini 3D.

## 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)

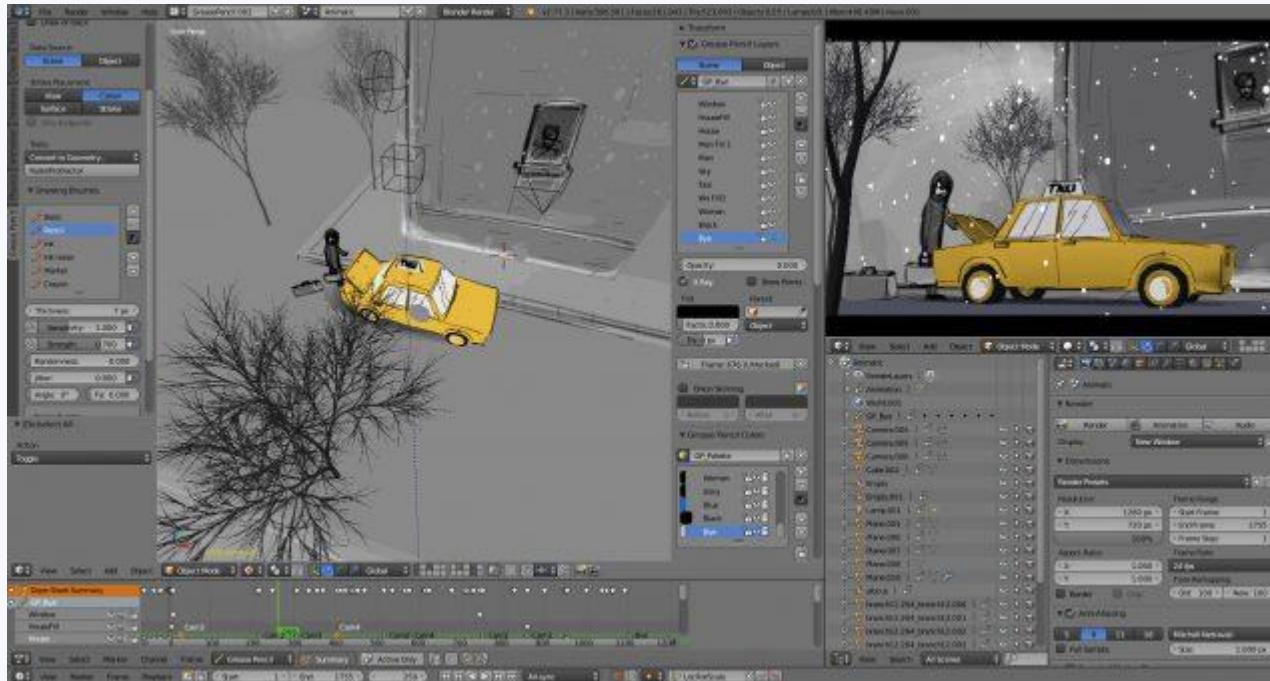
# Aplicații prelucrare imagini vectoriale



# Aplicații prelucrare imagini 3D

- folosit pentru a modela obiecte 3D, a defini suprafete, a compune scene și a exporta imaginea rezultată;
- În etapa de modelare (en: *modeling*), se creează forma obiectului; în etapa ulterioară, de definire a suprafetei (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)

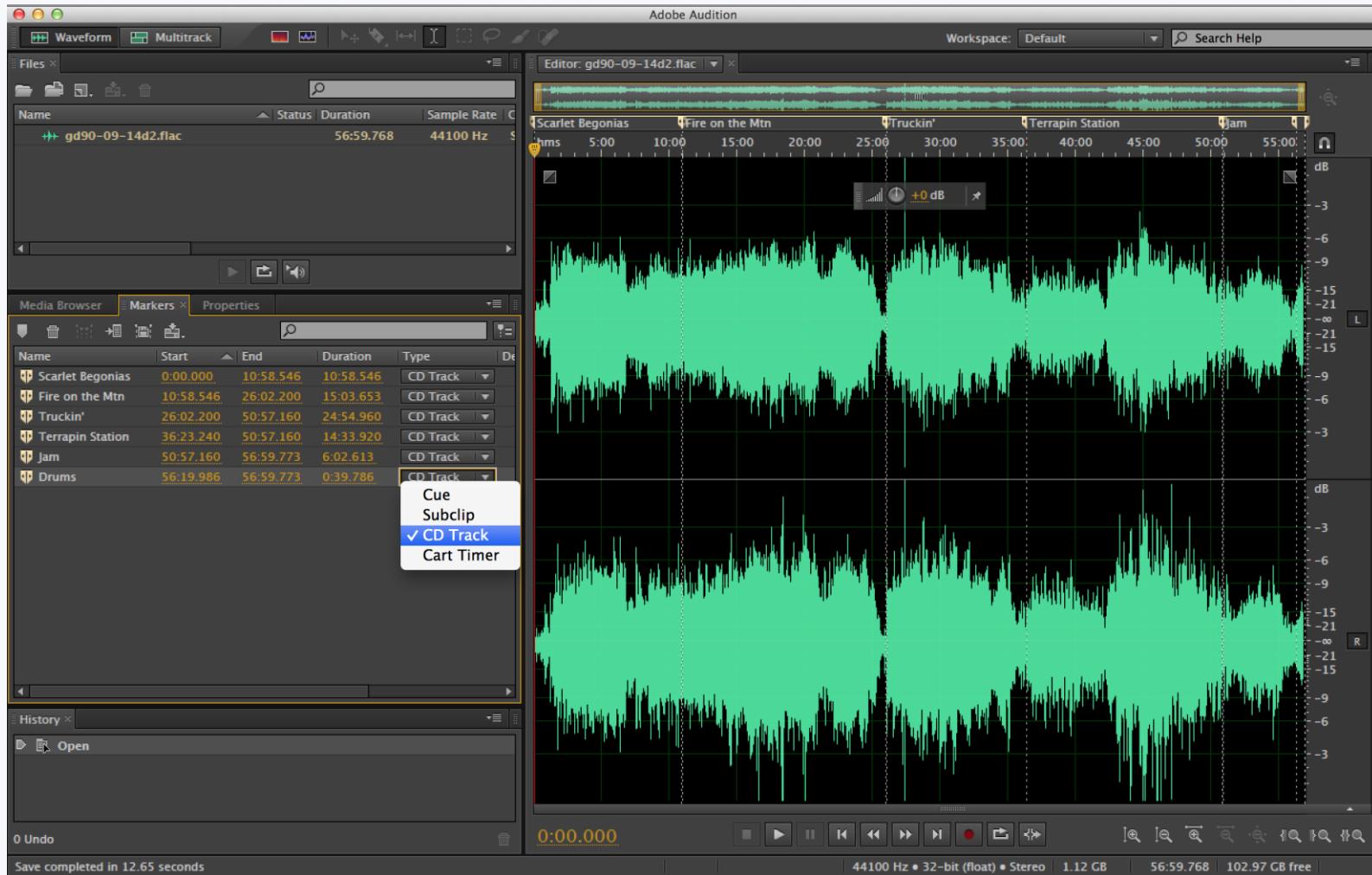
## Sunet eşantionat

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

# Software specializat Sunet



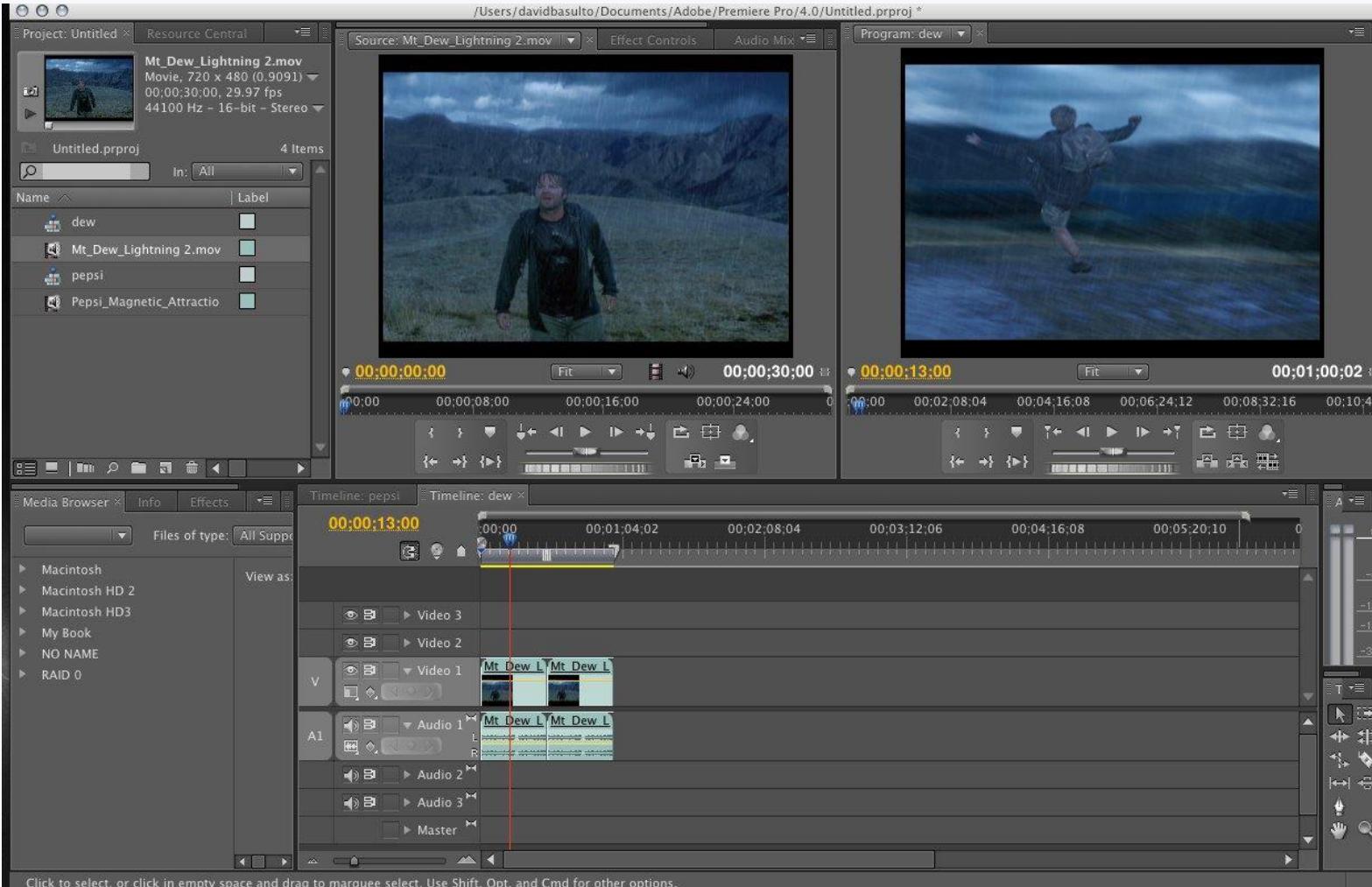
## Video

- 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 tranzitii 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..

# Software specializat

## 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.

Software specializat

## 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 projecța 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).

Software pentru dezvoltare multimedia

# Adobe Flash



# 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 – elemental <audio>
  - Video – elemental <video>

# HTML5 Games



[http://www.cuttherope.net/basic\\_standalone/game.html](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 type="text/javascript">
// 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](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`, `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*

Initializare: `var v = [];` sau `var v = [1, "Ion"]`

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_sfârșit)` – extrage un sub-vector

# Functii

Declarare:

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

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

Parametri:

Transmisi prin valoare

Accesibili prin *arguments*

Apel - nume\_functie / 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; ob.clasa = 2;

# 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(ume) { this.ume = 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.ume); };*

Folosit și pentru implementarea moștenirii:

```
function Student() { this.facultate = "CSIE"; }
```

```
Student.prototype = new Persoana("-");
```

```
var s = new Student(); s.afisare();
```

# DOM - Structura

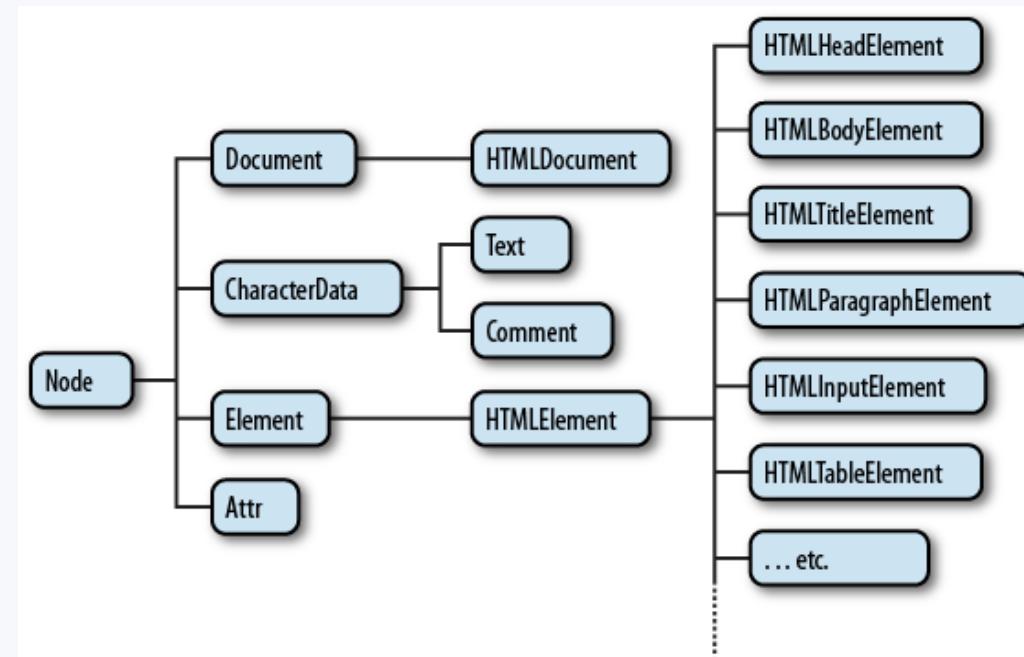
Arbore construit din obiecte JavaScript

Fiecare nod:

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

Rădăcina: *document.documentElement*

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



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

# Regăsire noduri

Pe bază de identificator

```
elem = document.getElementById(identificator)
```

Pe bază de selector CSS

```
elem = element.querySelector("selector CSS");
```

```
lista = element.querySelectorAll("selector CSS");
```

Alte metode

```
lista = element.getElementsByClassName("clasa CSS");
```

```
lista = element.getElementsByTagName("tag HTML");
```

# Manipulare noduri

Construire nod:

```
elem = document.createElement("tag HTML");
```

Accesare atribut:

elem.numeAtribut – accesare valoare atribut individual

elem.attributes – colecția de atribut

Accesare atribut 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 atribut 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*, *mouseup*, *mousedown*, *click*, *dblclick*

# Programare execuție funcție

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

Programare pornire:

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

Oprire:

*clearTimeout(id);*

B) Execuție repetată la un interval de timp fix

Programare pornire:

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

Oprire:

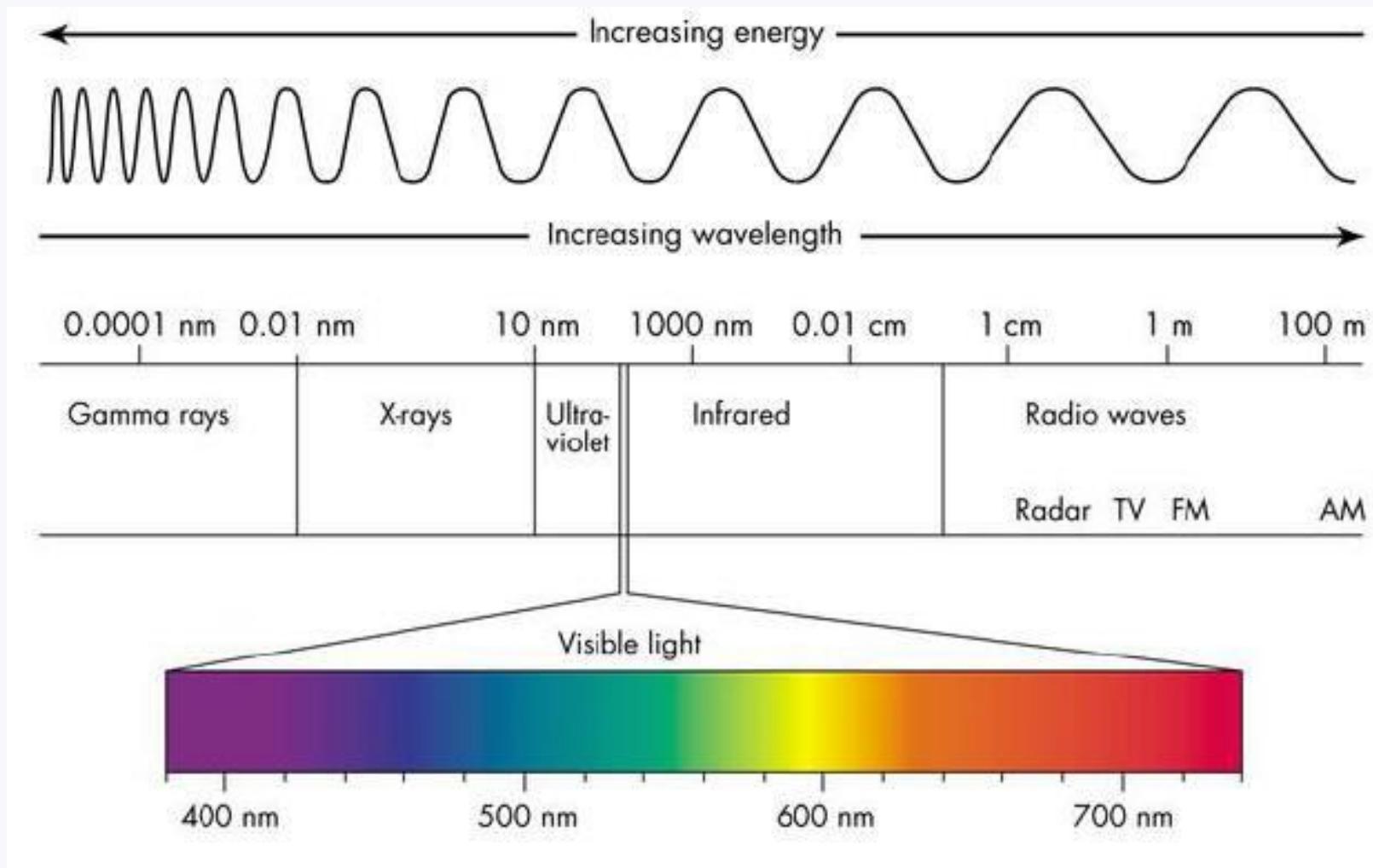
*clearInterval(id);*

Observație: duratele sunt exprimate în milisecunde

Imaginea

# Imagini digitale

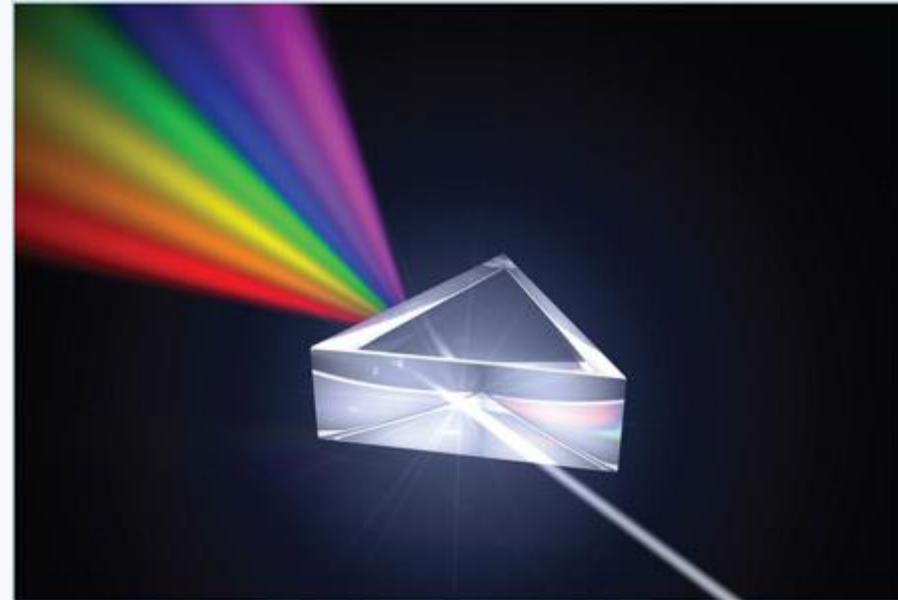
- O **imagine** este o reprezentare bidimensională sau tridimensională a unei persoane, obiect sau scenă din lumea naturală.
- O **imagine digitală** este o reprezentare numerică a unei imagini bidimensionale.



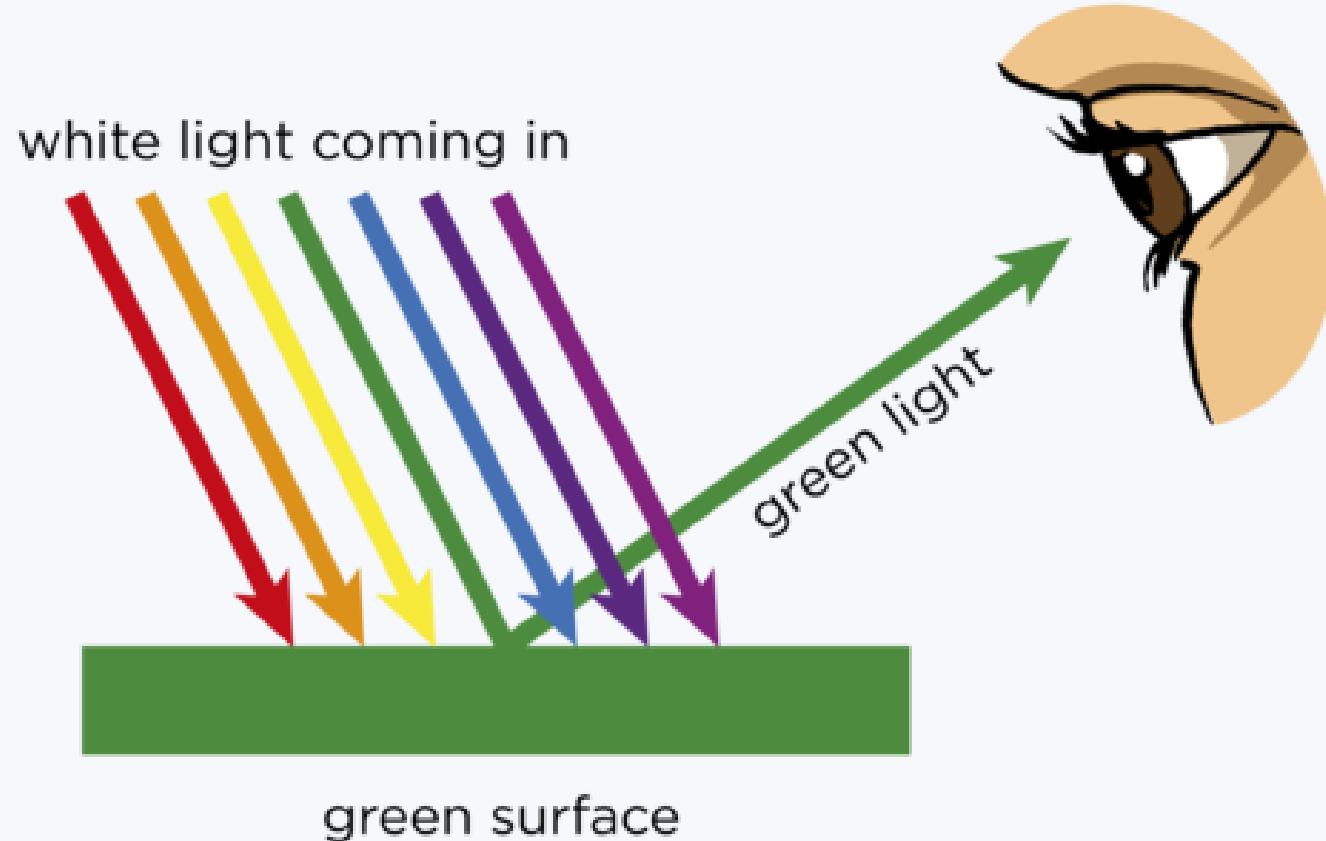
## Lumina

- ne referim la lumina soarelui naturală ca lumină albă, deoarece ochiul uman o percepă 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.

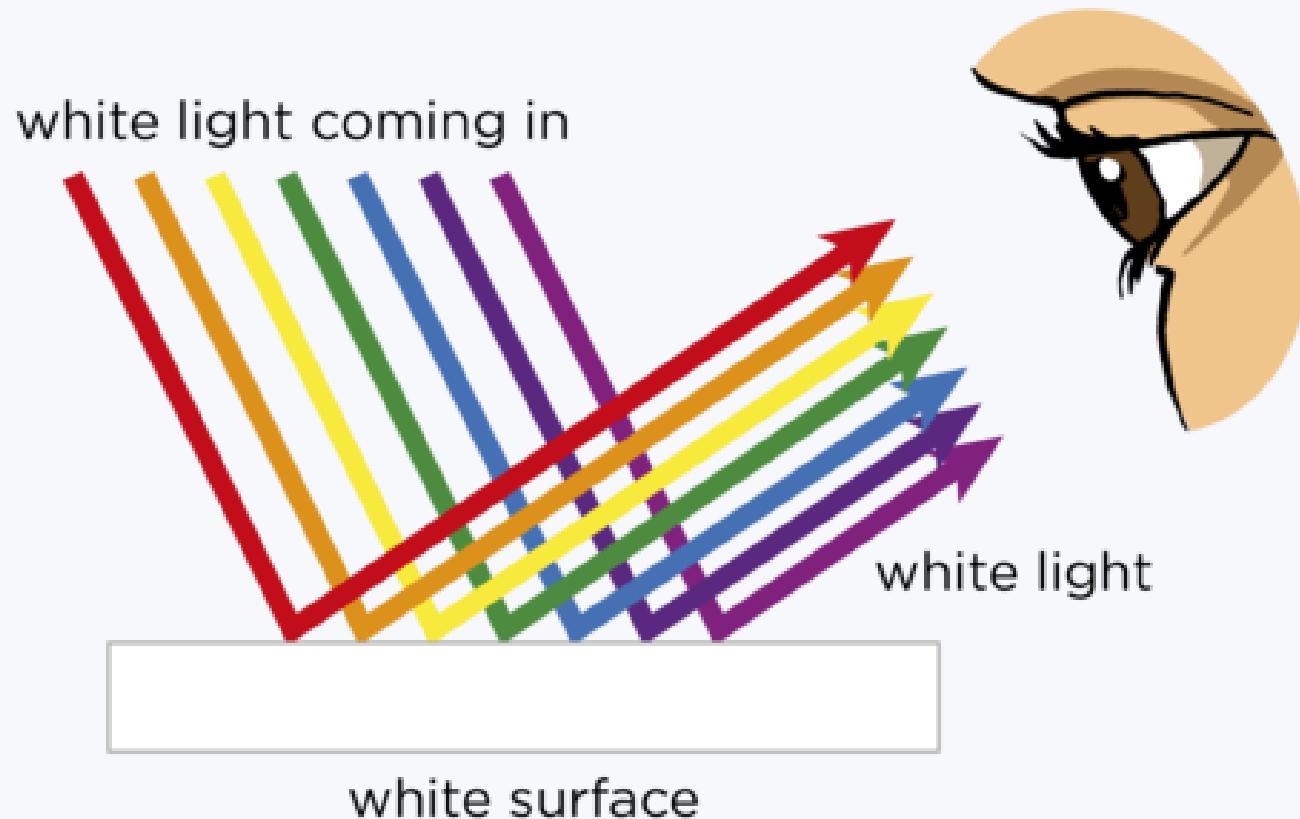
# Imaginea Lumina



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



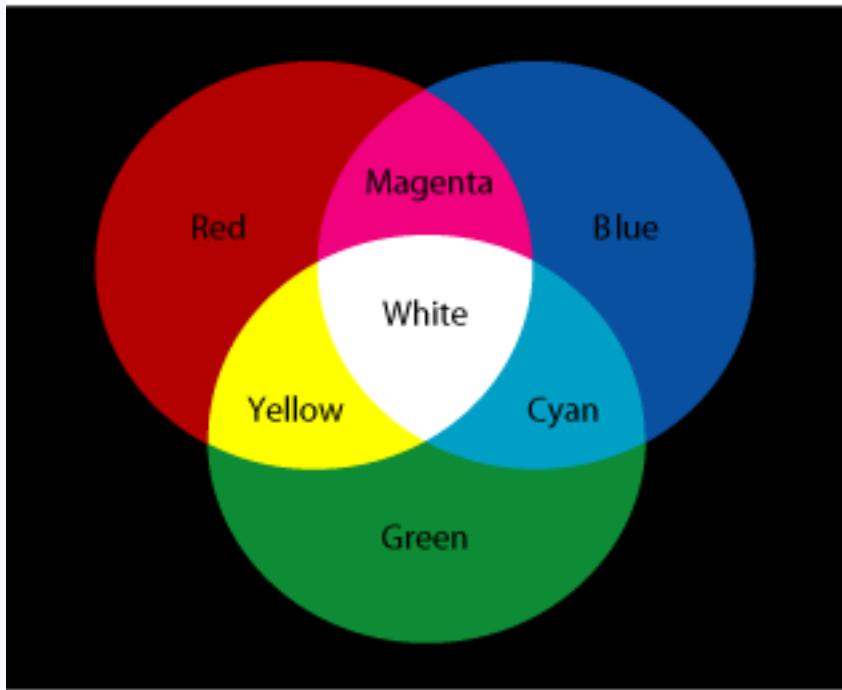
O suprafață verde absoarbe toate culorile, cu excepția verdelui, pe care îl reflectă.



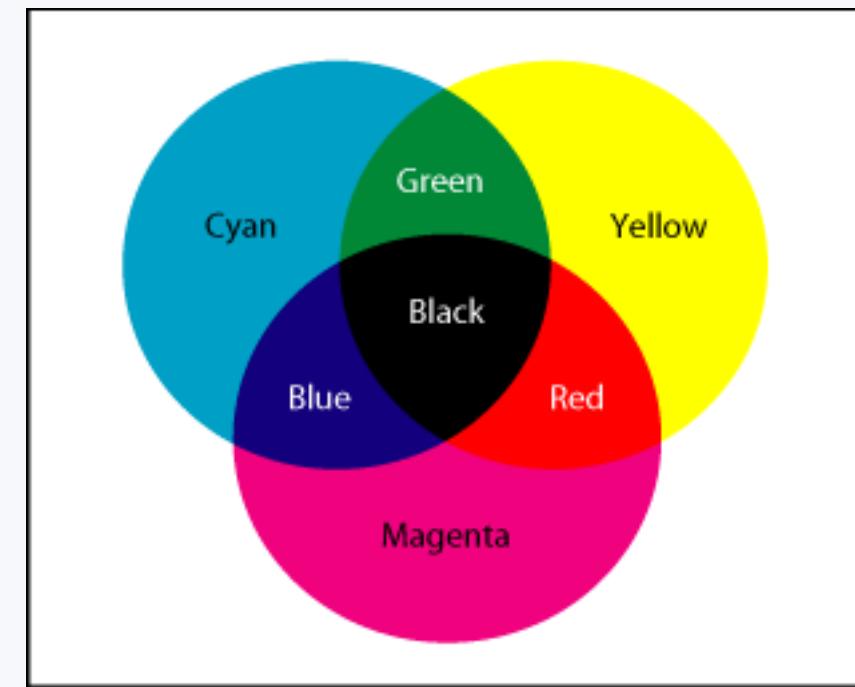
## Lumina

- 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 (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**

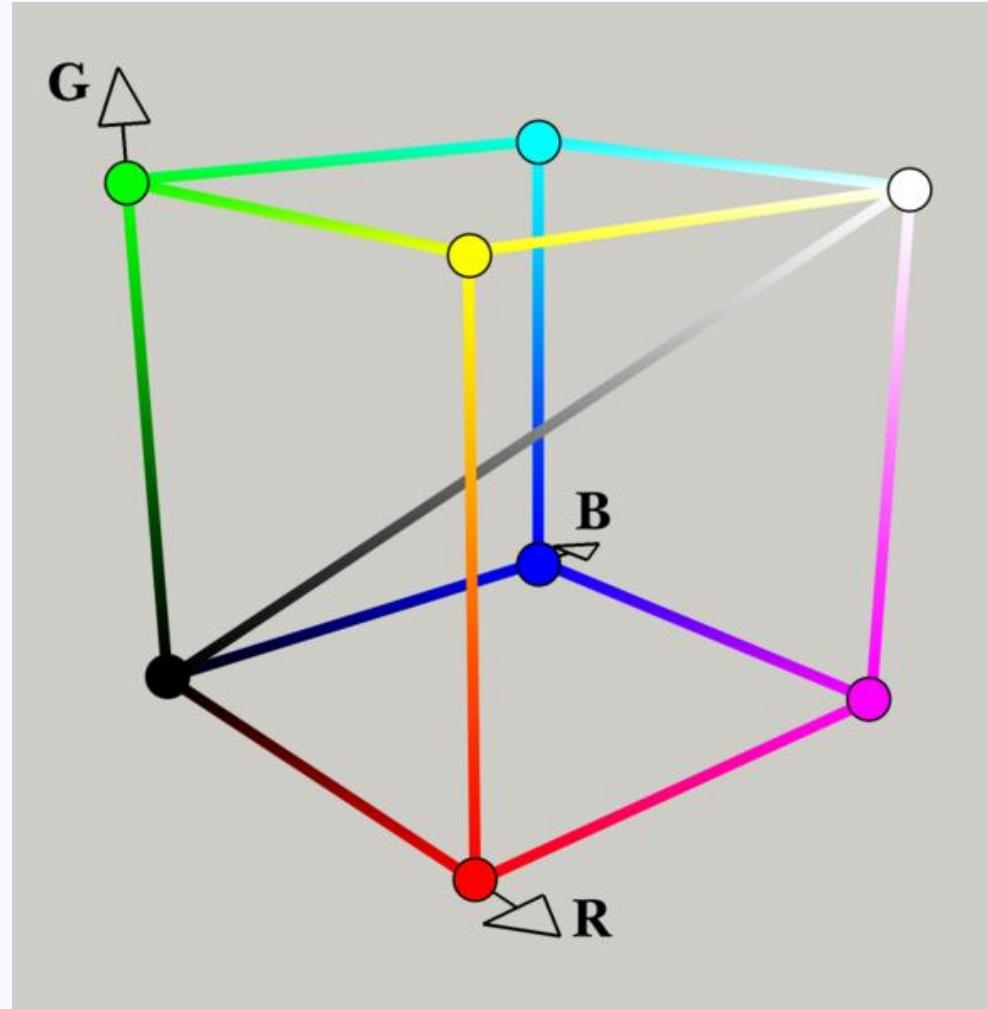
## Modelul RGB

- 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.

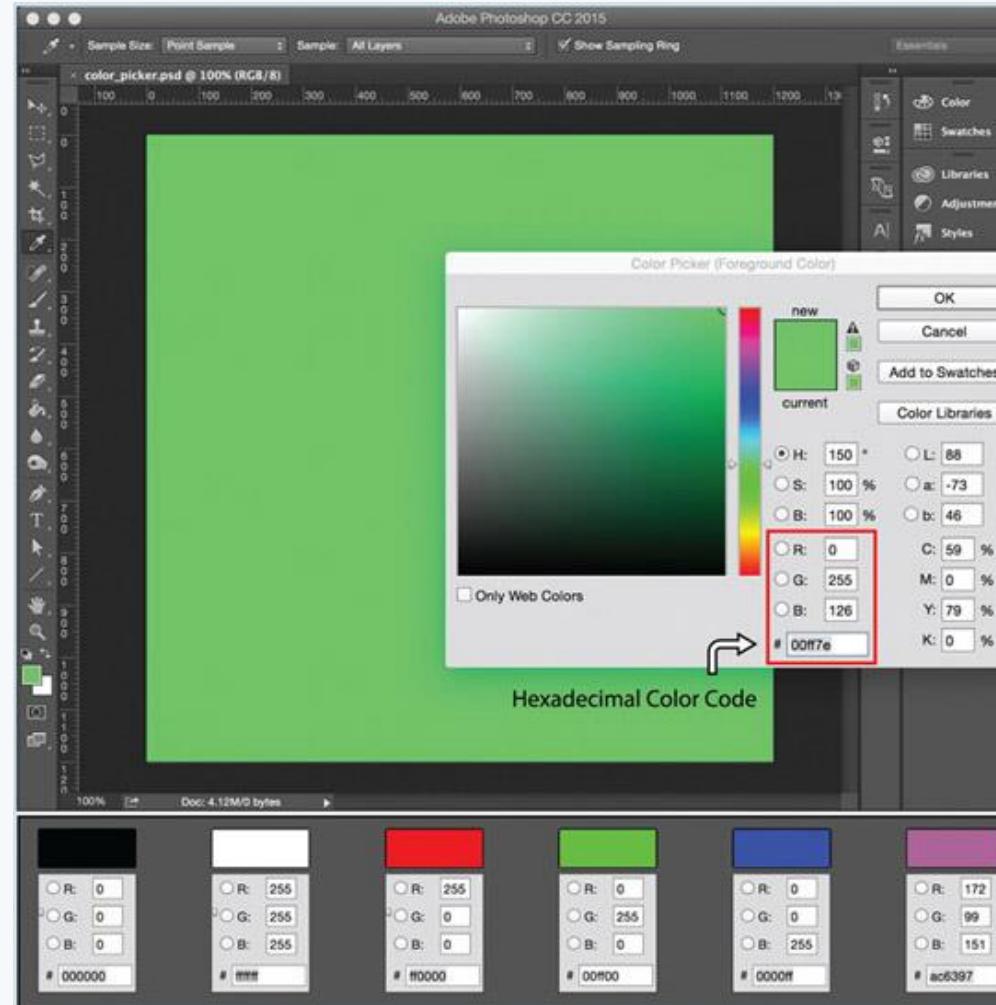
## Modelul RGB

- 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 componente 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 percep ca galben.

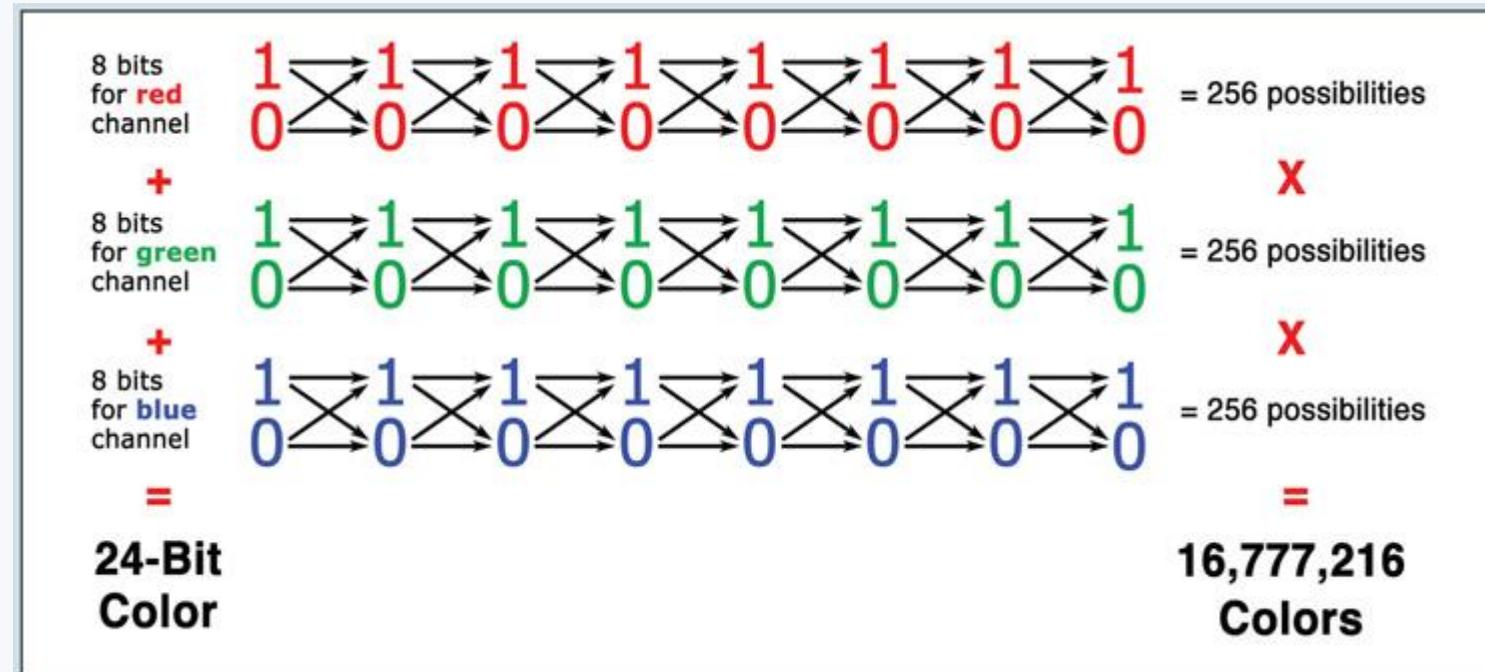
# Modelul RGB



# Adobe Photoshop – Modelul RGB



# Modelul RGB



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

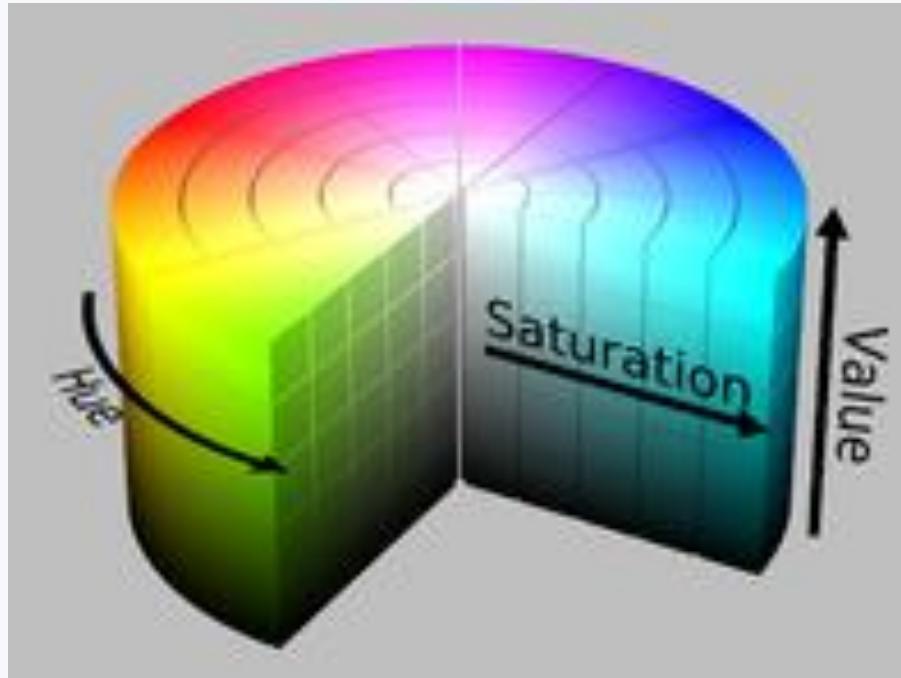
## Modelul RGB

- 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 carteziană (cub)

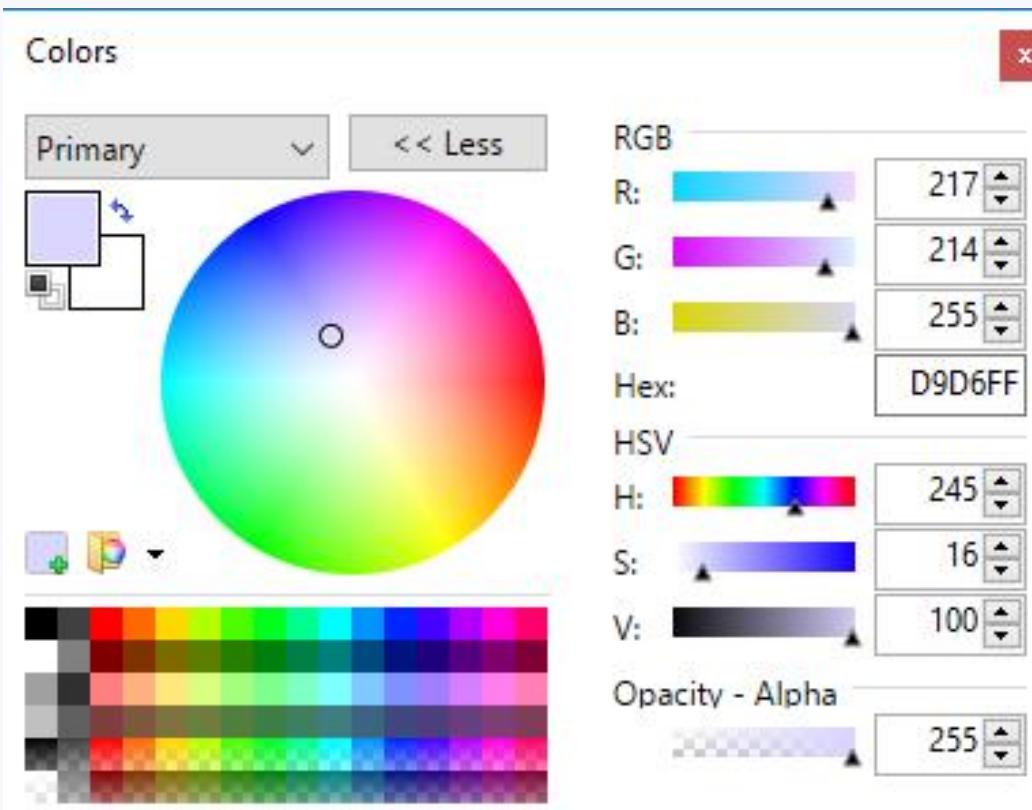
# HSL - Hue, Saturation and Lightness Color Model



- 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 subtractive

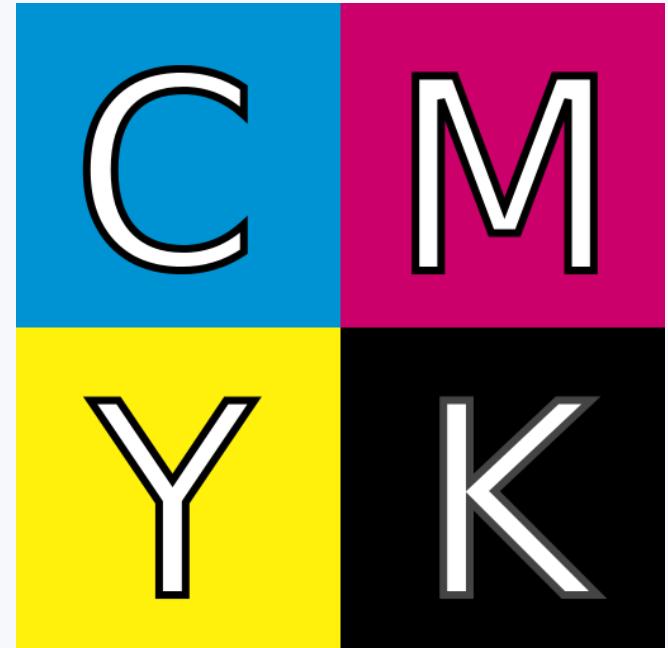
- Combinarea culorilor în procesul de imprimare utilizează metoda **subtractivă** [1].
- Modelul **subtractiv** explică cum este posibil ca prin combinarea unui număr limitat de pigmenți sau coloranți naturali să se obțină o paletă largă de culori. Adăugarea fiecărei culori determină reducerea parțială 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](https://en.wikipedia.org/wiki/Subtractive_color)

[2] [http://www.worqx.com/color/color\\_systems.htm](http://www.worqx.com/color/color_systems.htm)

## Modelul CMYK

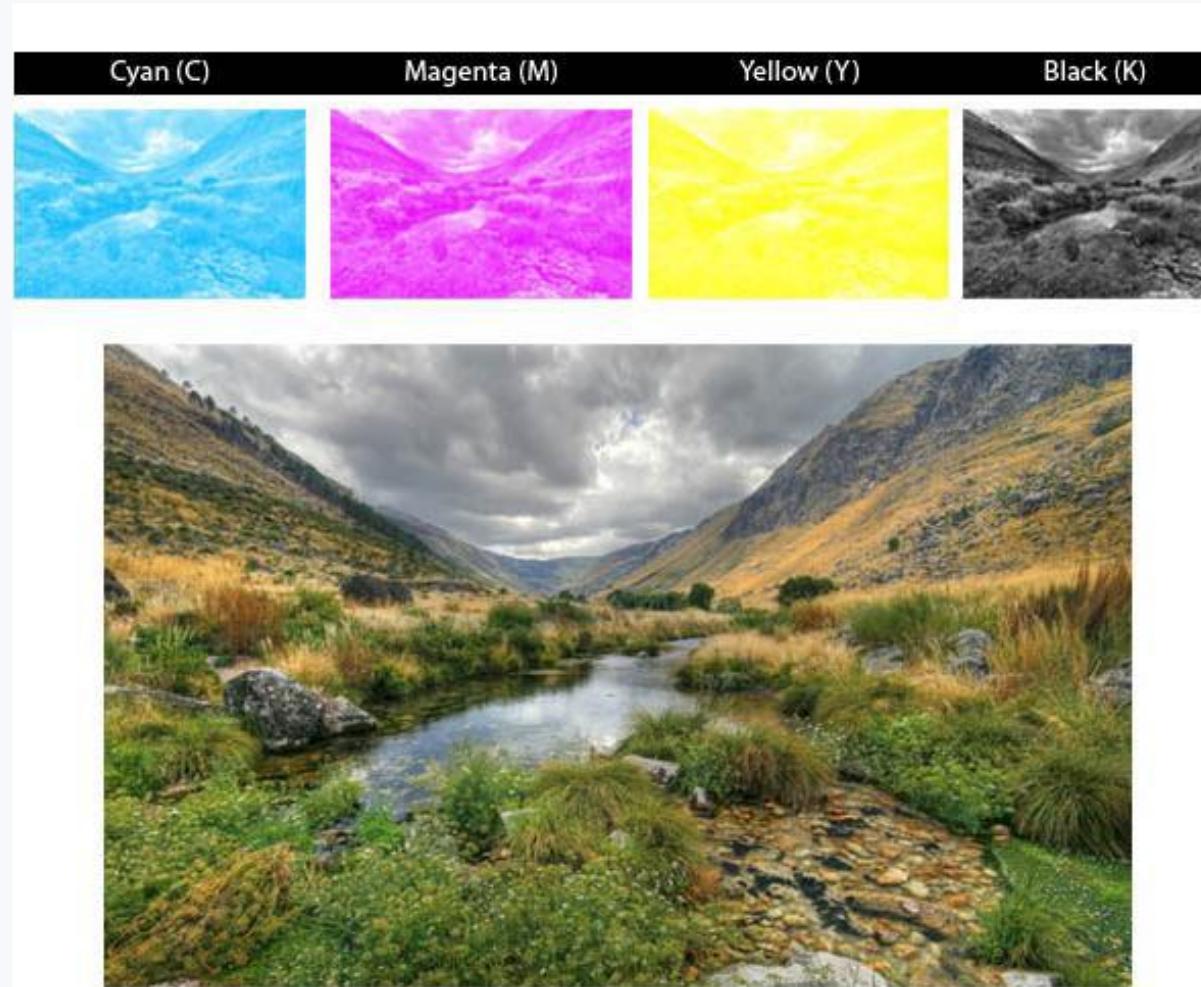
- Modelul de culoare CMYK este utilizat în procesul de imprimare[1].
- CMYK se referă la cele patru cerneluri utilizate în majoritatea imprimărilor color: cyan, magenta, yellow și key (black) [1].
- Deși variază în funcție de tipografie, producător, imprimantă, cerneala este de obicei aplicată în ordinea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



# Modelul CMYK

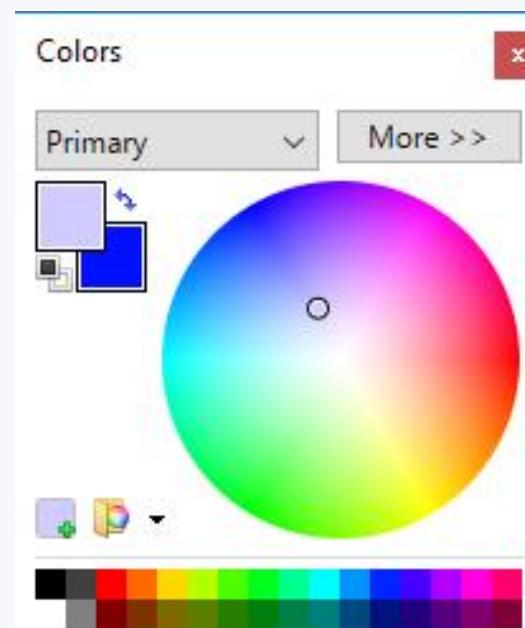


# Modelul CMYK

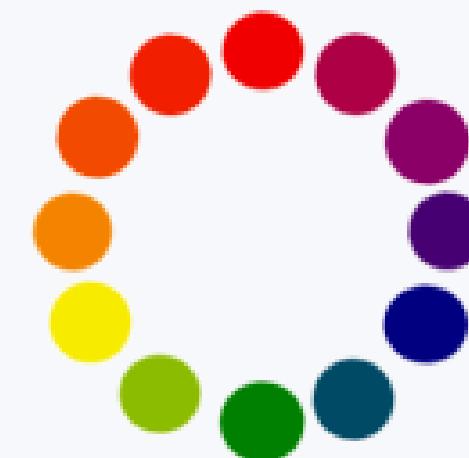


# Cercul culorilor (en: Color Wheel)

- 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>



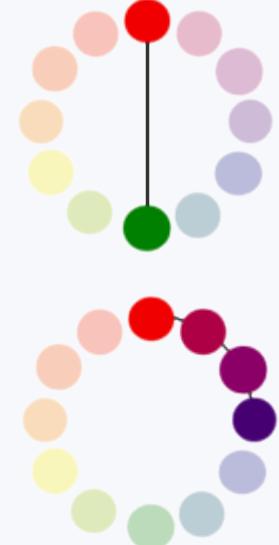
# Cercul culorilor (en: Color Wheel)

- **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].



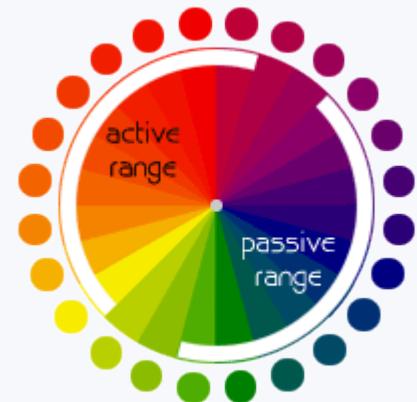
# Active & Passive Colors

- 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

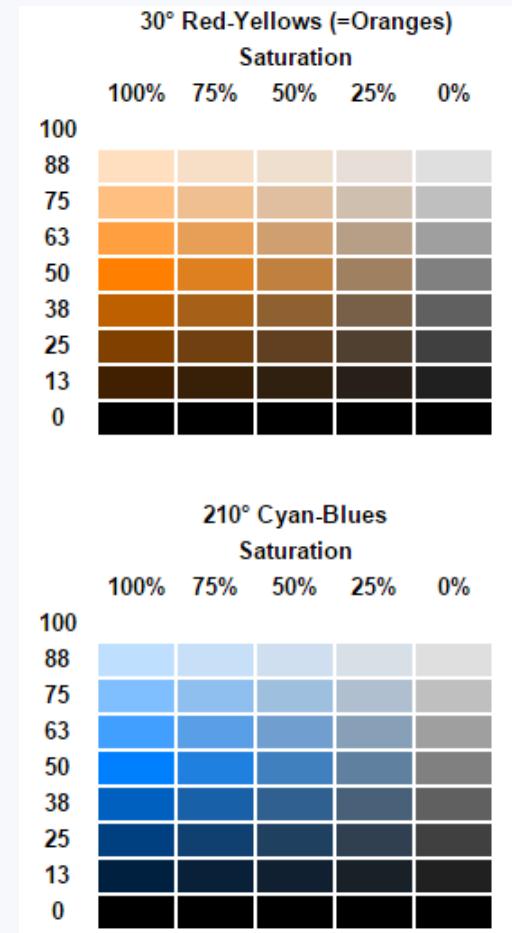
```
<h1 style="background-color:rgb(255, 99, 71);">...</h1>
```

# 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>
```

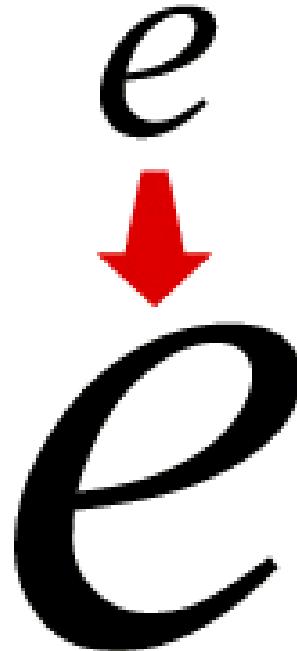
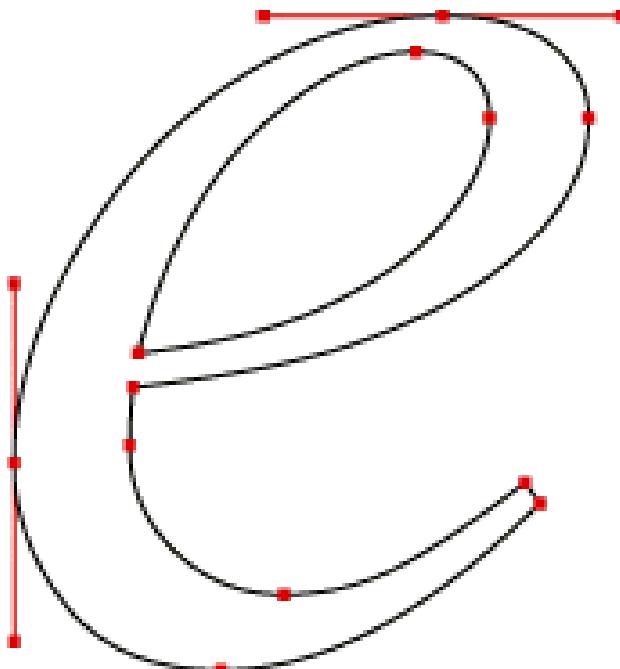


# Grafică pe computer 2-D

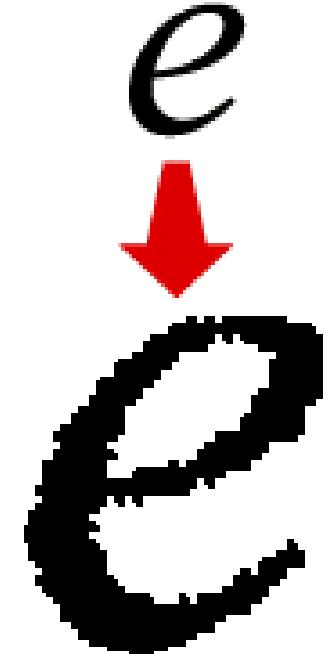
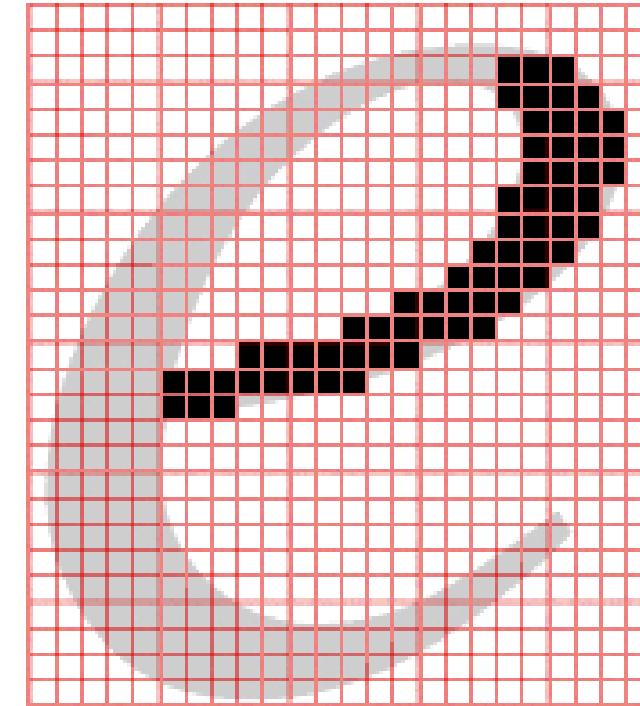
- Calculatoarele pot fi utilizate pentru a crea imagini 2D (lățime și înăltime) sau 3D (lățime, înăltime și adâncime).
- Există două tipuri principale de grafică computerizată 2D:
  - **bitmap** - potrivită pentru imagini cu detalii fine, cum ar fi picturi sau fotografii. Numită și **grafică raster**. 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

VECTOR GRAPHICS



BITMAPPED (RASTER) GRAPHICS

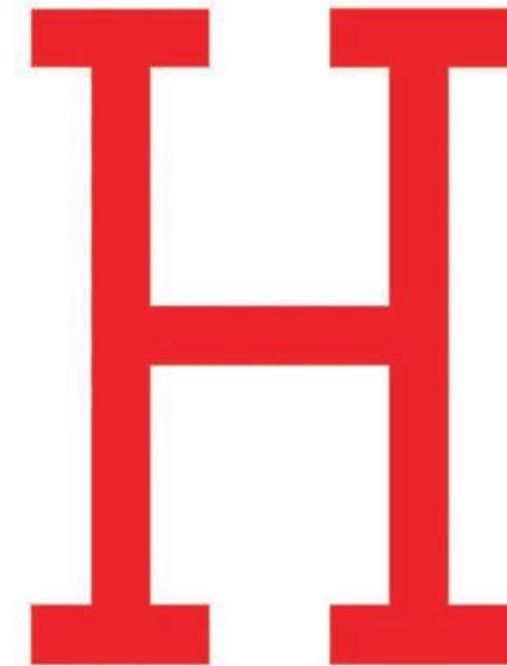


# Grafică raster

# 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 ocasional rectangulară) de culoare, reprezentând un singur punct din imagine [2].

0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	1	1	1	0
0	0	1	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0
0	0	1	1	1	1	1	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0
0	1	1	1	1	0	0	1	1	1	0
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 eliberați le reducerea dimensiunilor unei imagini raster. Lățimea și înălțimea unei imagini raster sunt determinate de câte pixeli conține fiecare rând și coloană.

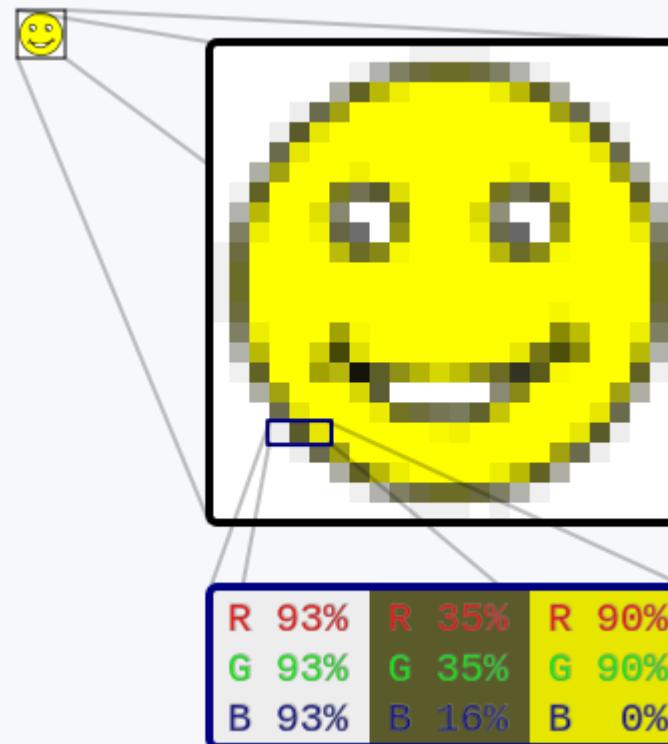
# Mozaic

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 imaginii unei imagini raster și se referă direct la dimensiunea și cantitatea de pixeli pe care o conține imaginea.
- **Dimensiuni în pixeli**
  - descrieți dimensiunea unei imagini raster, exprimată ca numărul de pixeli de-a lungul axei x (lățime) de 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, multiplicați dimensiunile orizontale și verticale ale pixelilor.
  - 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-bitî (256 culori), 24 bitî (16.8 milioane de culori)
  - Un desen folosind 16 culori distințe, de exemplu, ar necesita doar 4 biți pe pixel. Utilizarea unui cod cu o adâncime de biți mai mare produce un fișier 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:

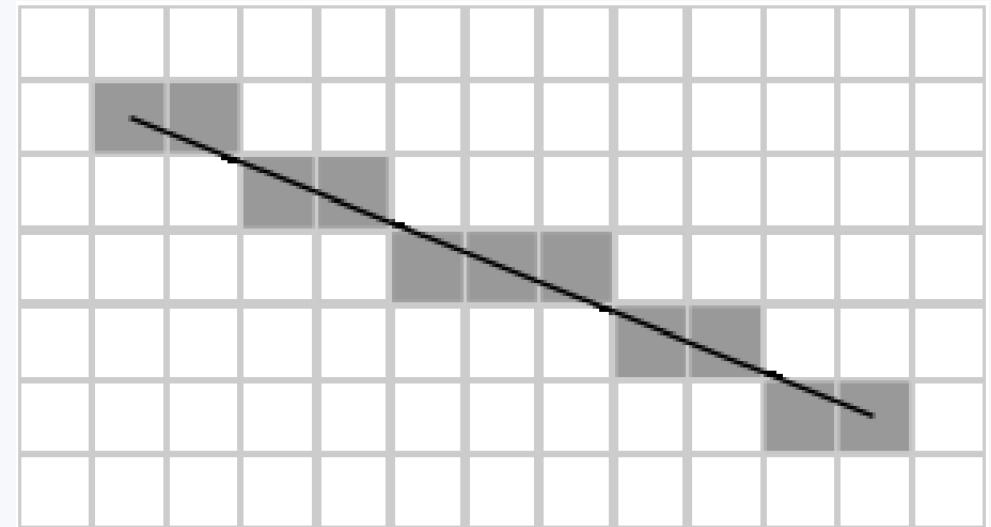
1. Salvați imaginile de mai jos și analizați modul în care reacționează la modificarea dimensiunilor.
2. Care dintre cele două imagini este de tip raster?



# Desenare elemente grafice

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

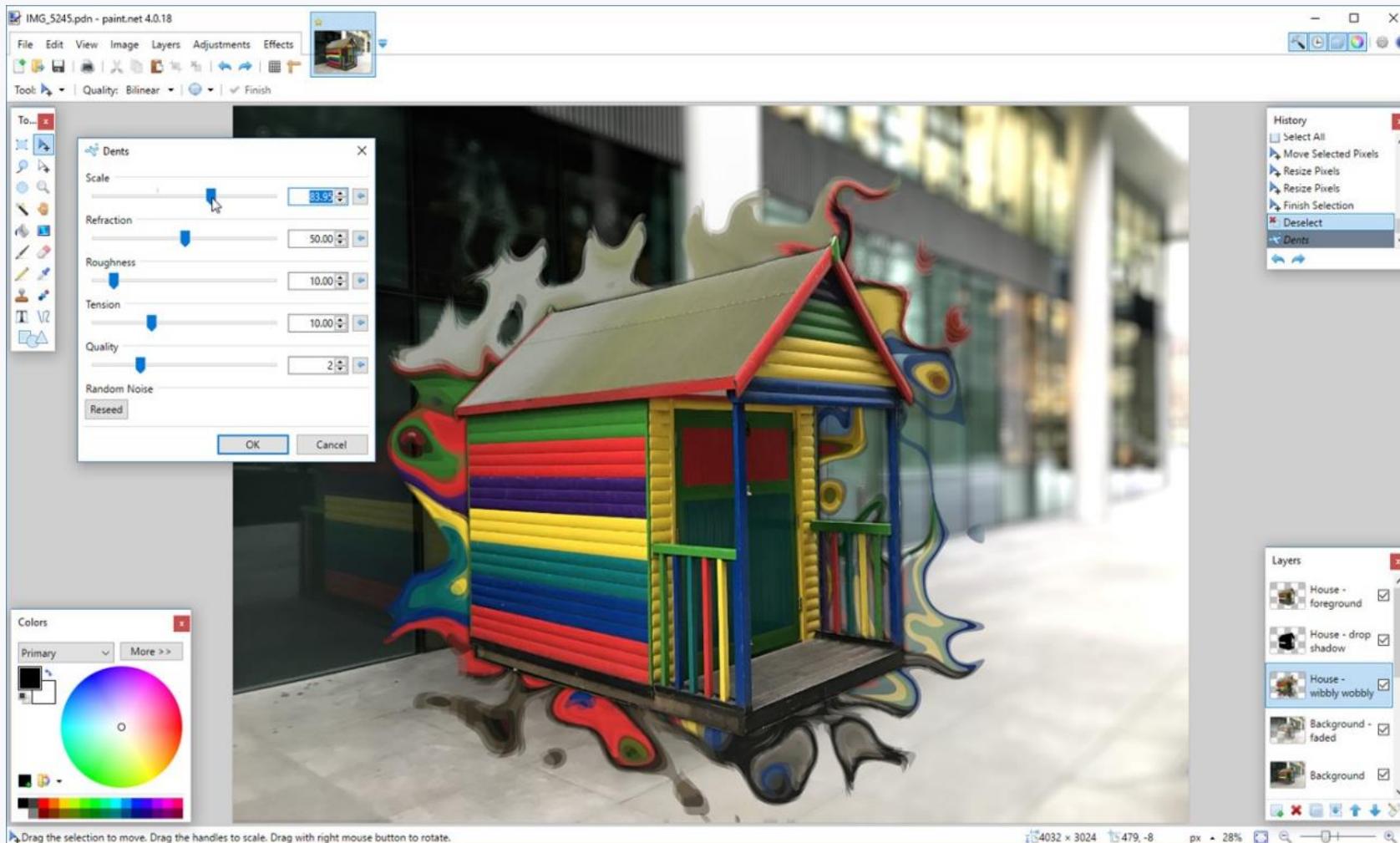
Exemplu: segment de dreaptă  $(x_1, y_1) \rightarrow (x_2, y_2)$



## 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 imaginea 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/>

# Grafică raster 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

- GIF (Graphics Interchange Format) - \*.gif
  - maxim 256 de culori și transparentă (pixeli transparenti);
  - 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 transparentă (nu permite definirea de pixeli transparenti).
  - format de compresie cu pierderi
  - este folosit cel mai adesea pentru fotografii

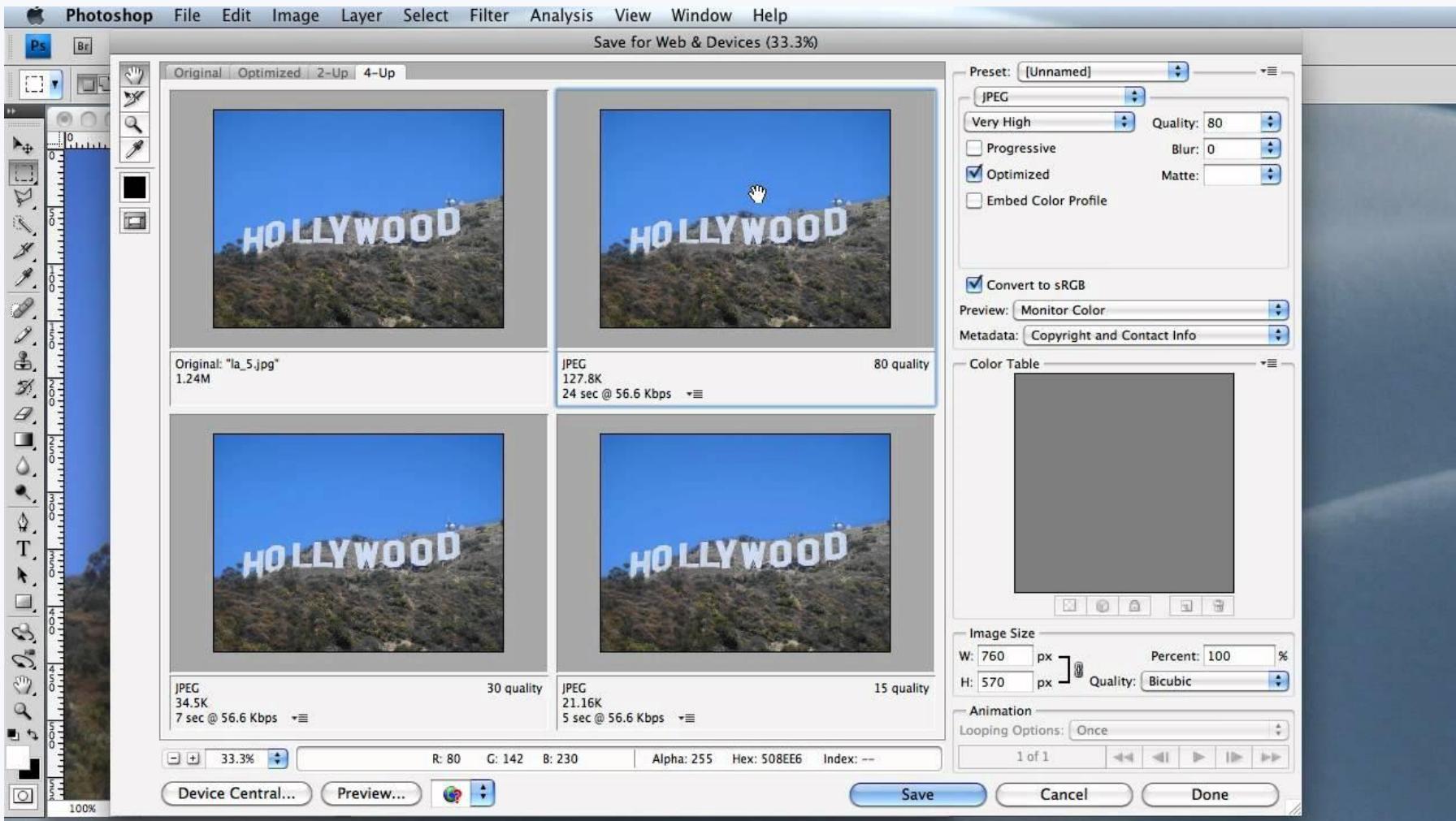
## Formate



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

# Grafică raster

# Formate

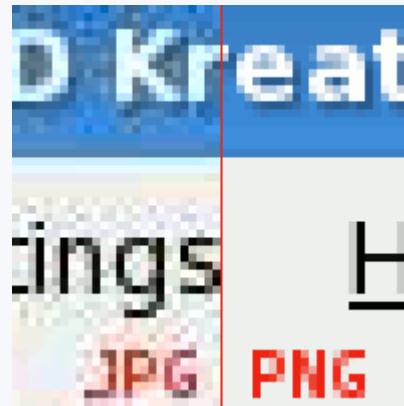


Photoshop – meniul Save for Web

## 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 transparentă; 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 aceleasi 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](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.

Grafică raster

## Imagini de dimensiuni mari

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



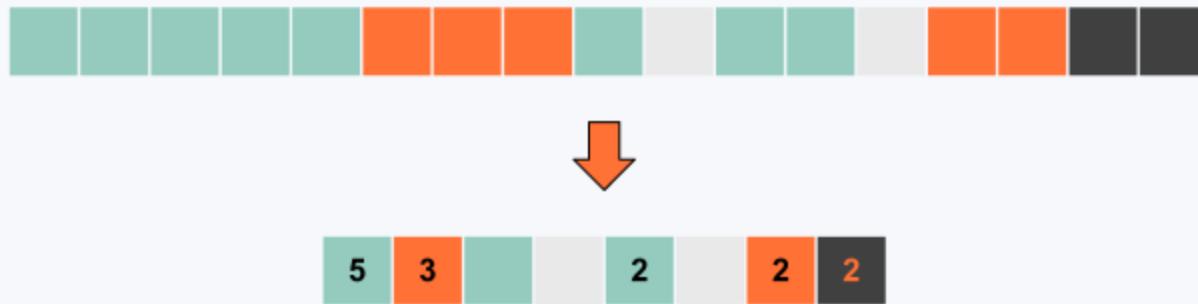
# Compresie

- 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

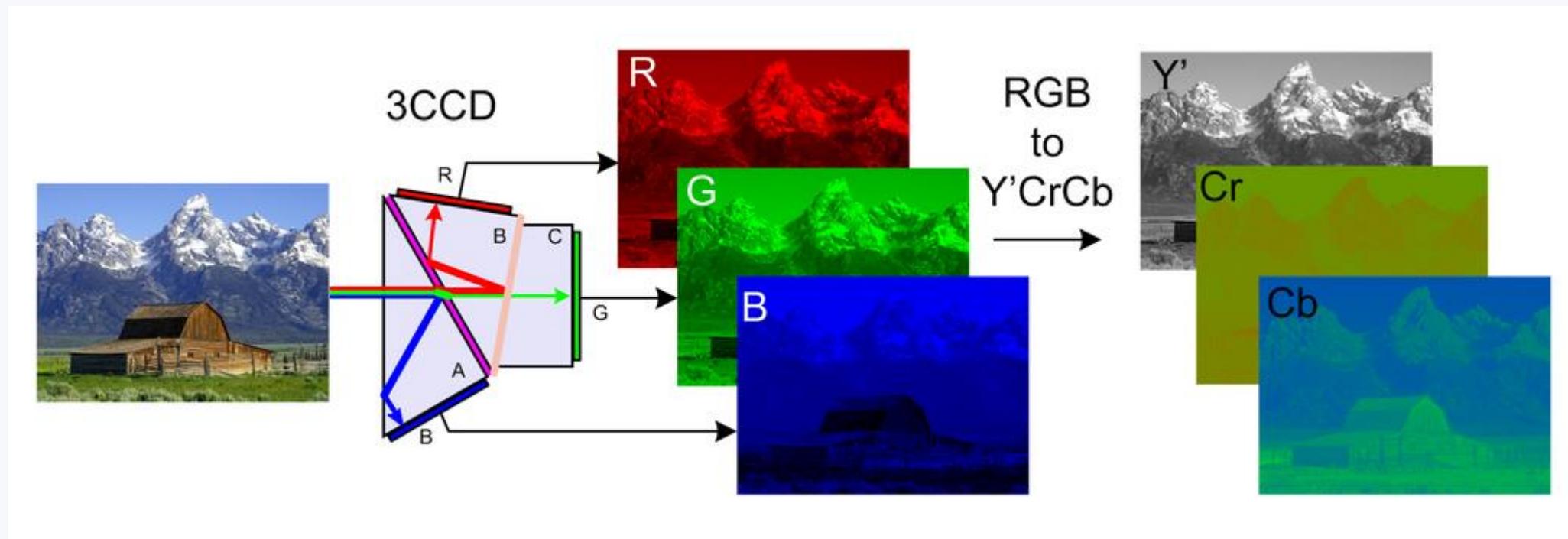
- lossy data compression algorithm

- 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'CrCb, consisting of one luma component ( $Y'$ ), representing brightness, and two chroma components, (CB and CR), representing color. This step is sometimes skipped.



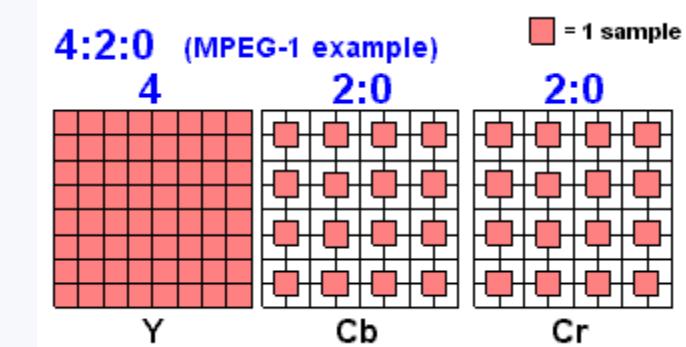
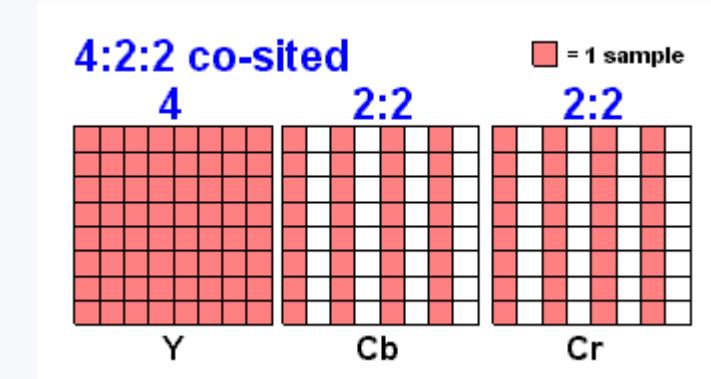
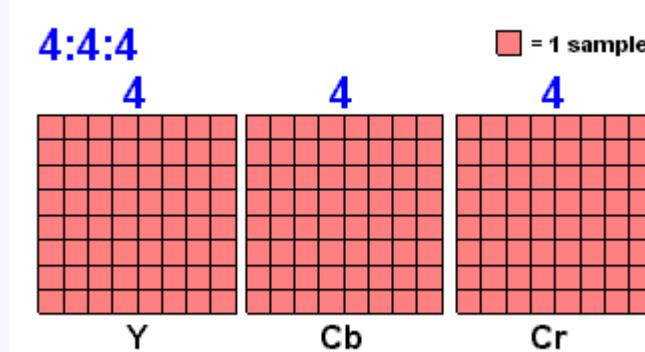
# JPEG Compression - Step 1 - Color space transformation

- 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 C<sub>b</sub> and C<sub>r</sub> components). Using this knowledge, encoders can be designed to compress images more efficiently.
- The transformation into the Y'C<sub>B</sub>C<sub>R</sub> color model enables the next usual step, which is to reduce the spatial resolution of the C<sub>b</sub> and C<sub>r</sub> components.

# JPEG Compression – Step 2 - Downsampling



## JPEG Compression – Step 2 - Downsampling

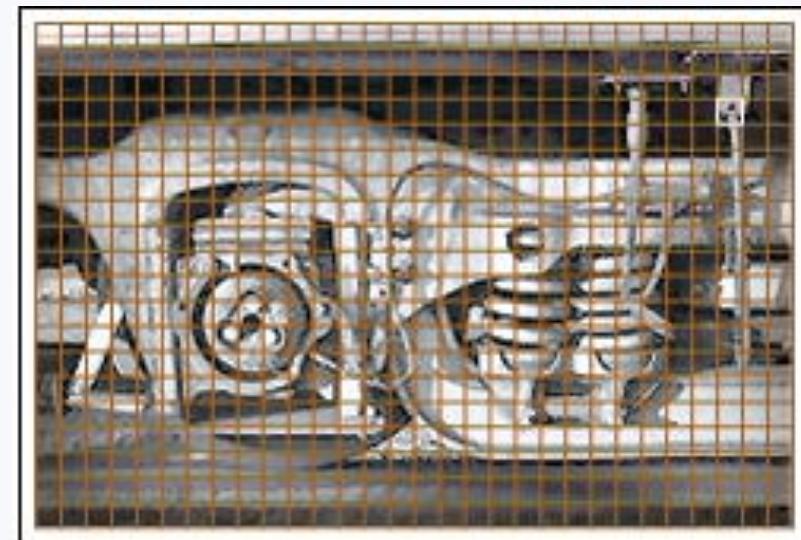
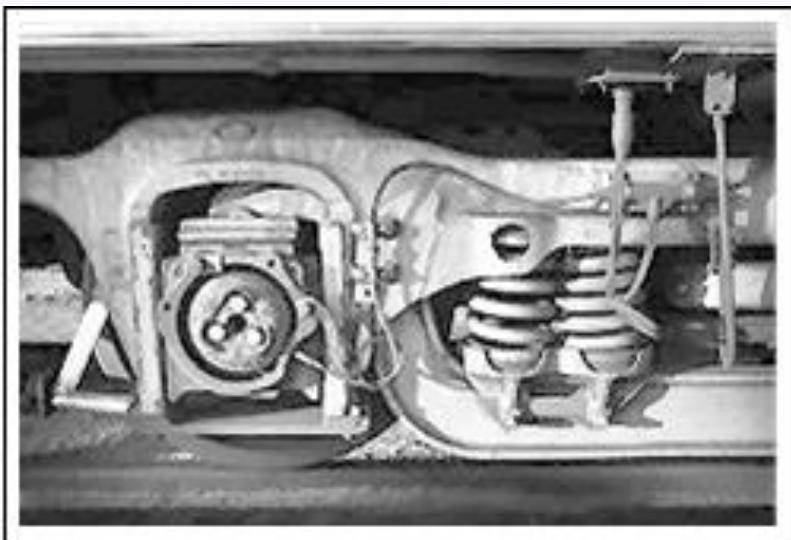
- The ratios at which the downsampling is ordinarily done for JPEG images are:
  - 4:4:4 - no downsampling,
  - 4:2:2 - reduction by a factor of 2 in the horizontal direction
  - 4:2:0 - reduction by a factor of 2 in both the horizontal and vertical directions (most common).
- For the rest of the compression process, Y', Cb and Cr are processed separately and in a very similar manner.

## JPEG Compression – Step 3 - Block splitting

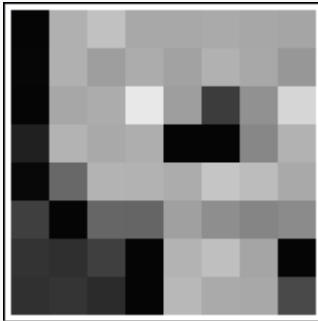
- Each channel must be split into  $8 \times 8$  blocks
- Depending on chroma subsampling, this yields Minimum Coded Unit (MCU) blocks of size  $8 \times 8$  (4:4:4 – no subsampling),  $16 \times 8$  (4:2:2), or most commonly  $16 \times 16$  (4:2:0).

# 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  $160 \times 240 = 8*20 \times 8*30$ . So this step creates  $20 \times 30 = 600$  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  $8 \times 8$  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 simplify 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  $8 \times 8$  block.
- The DCT is a product  $C = U B U^T$  where  $B$  is an  $8 \times 8$  block from the preprocessed image and  $U$  is a special  $8 \times 8$  matrix.
- Loosely speaking, the DCT tends to push most of the high intensity information (larger values) in the  $8 \times 8$  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

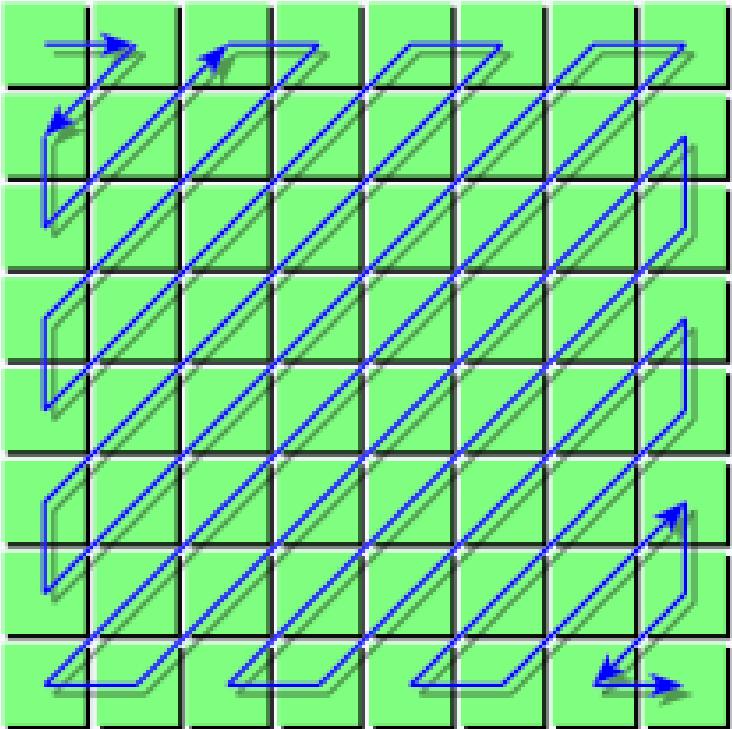
16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

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 reversible steps. The ability to recover the original image is lost.

# HTML5 Image Element

- afişarea 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)
});
image.setAttribute('src', "mask.png");
```

# HTML5 Canvas Element

- Poate fi folosit pentru a desena grafice, a face compozitii foto sau chiar a efectua animatii.
- Declararea unui element de tip canvas :

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

- API: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas>
- DOM interface: HTMLCanvasElement

# HTML5 Canvas Element - Metode

- `getContext(contextType, contextAttributes);` – returns a drawing context on the canvas, or null if the context identifier is not supported;

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

- `contextType`: DOMString containing the context identifier defining the drawing context associated to the canvas.

# HTML5 Canvas Element - Metode

- possible values for contextType:
  - "2d", leading to the creation of a `CanvasRenderingContext2D` object representing a two-dimensional rendering context.
  - "webgl" which will create a `WebGLRenderingContext` object representing a three-dimensional rendering context. This context is only available on browsers that implement WebGL version 1 (OpenGL ES 2.0).
  - "bitmaprenderer" which will create a `ImageBitmapRenderingContext` which only provides functionality to replace the content of the canvas with a given `ImageBitmap`.

# HTML5 Canvas Element - Interfața CanvasRenderingContext2D

- used for drawing rectangles, text, images and other objects onto the canvas element. It provides the 2D rendering context for the drawing surface of a <canvas> element.
- API: [developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D)
- Drawing rectangles:
  - clearRect() - sets all pixels in the rectangle defined by starting point (x, y) and size (width, height) to transparent black, erasing any previously drawn content.
  - fillRect() - draws a filled rectangle at (x, y) position whose size is determined by width and height.
  - strokeRect() - paints a rectangle which has a starting point at (x, y) and has a width and an h height onto the canvas, using the current stroke style.

# HTML5 Canvas Element - Interfața CanvasRenderingContext2D

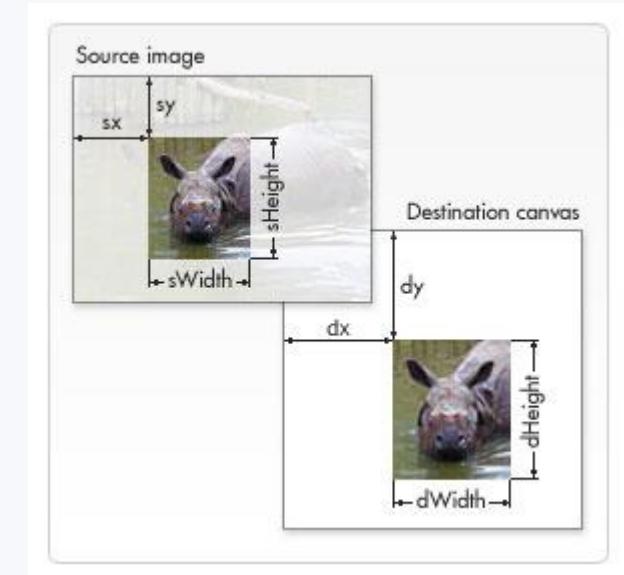
- Drawing text:
  - `fillText()` - draws (fills) a given text at the given (x,y) position;
  - `strokeText()` - draws (strokes) a given text at the given (x, y) position;
  - `measureText()` - returns a `TextMetrics` object.
- Drawing paths:
  - check [documentation](#)
- Line styles:
  - check [documentation](#)

# HTML5 Canvas Element - Interfața CanvasRenderingContext2D

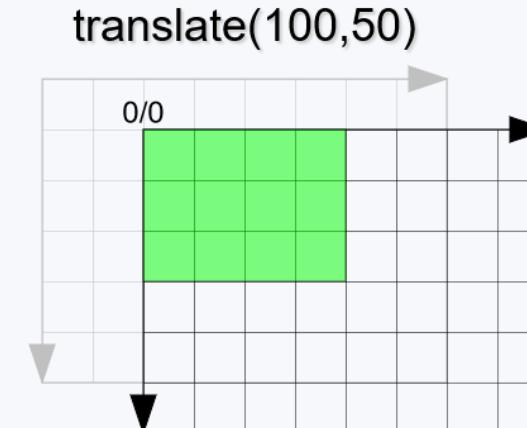
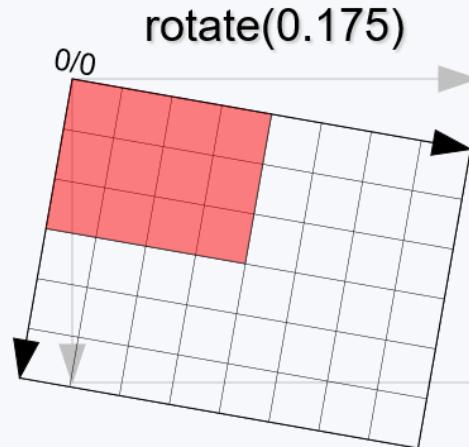
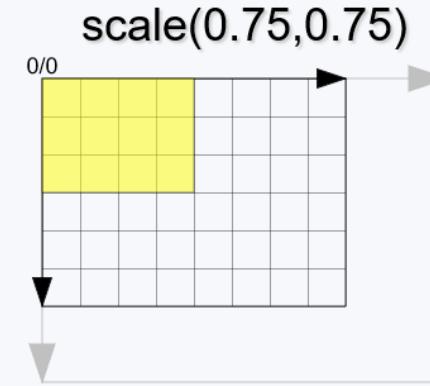
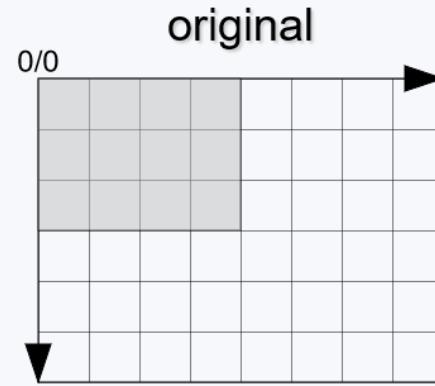
- Text styles:
  - check [documentation](#)
- Fill and stroke styles
  - check [documentation](#)
- Gradients and patterns
  - check [documentation](#)
- Shadows
  - check [documentation](#)

# HTML5 Canvas Element - Interfața CanvasRenderingContext2D

- Drawing images:
  - Without scaling
    - `void ctx.drawImage(image, dx, dy);`
  - With scaling
    - `void ctx.drawImage(image, dx, dy, dWidth, dHeight);`
    - `void ctx.drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight);`
  - API: [Mozilla Developer Network](#)

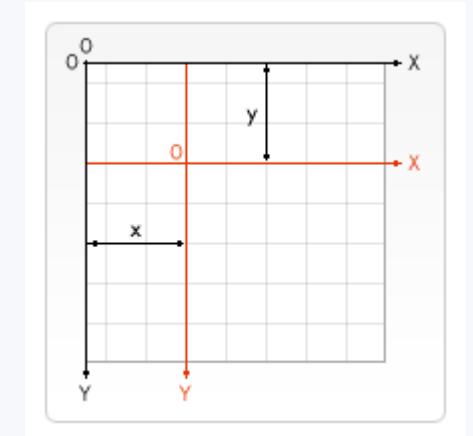


# HTML5 Canvas Element - Transformări



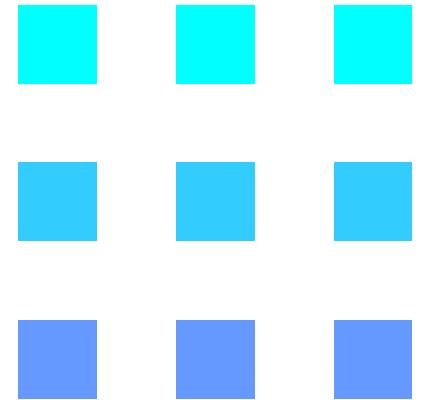
# HTML5 Canvas Element - CanvasRenderingContext2D Interface

- Translating
  - `translate(x, y)`
    - changes the origin.
    - x indicates the horizontal distance to move, and y indicates how far to move the grid vertically.
  - API: [Mozilla Developer Network](#)



# HTML5 Canvas Element - CanvasRenderingContext2D Interface

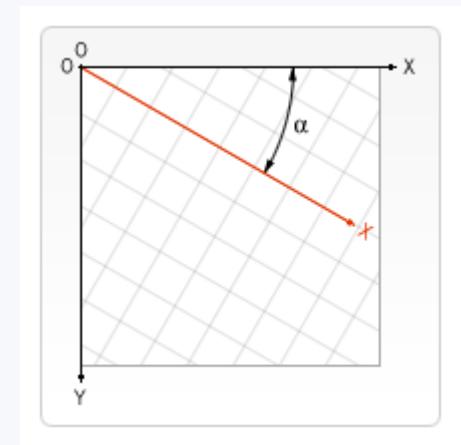
```
function draw() {  
    var ctx = document.getElementById('canvas').getContext('2d');  
    for (var i=0;i<3;i++) {  
        for (var j=0;j<3;j++) {  
            ctx.save();  
            ctx.fillStyle = 'rgb('+(51*i)+','+(255-51*i)+',255)';  
            ctx.translate(10+j*50,10+i*50);  
    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



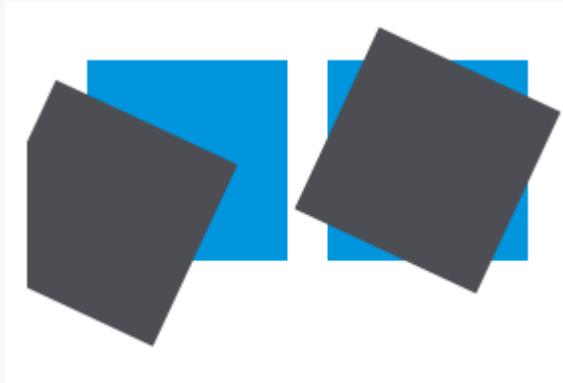
- Without the translate() method, all of the rectangles would be drawn at the same position (0,0). Using the translate() method we don't have to manually adjust the coordinates in the fillRect() function.
- JSFiddle: <https://jsfiddle.net/liviucotfas/9tfdegm7/>

# HTML5 Canvas Element - CanvasRenderingContext2D Interface

- Rotating
  - `rotate(angle)`
    - Rotates the canvas clockwise around the current origin by the angle number of radians.
    - The rotation center point is always the canvas origin, unless it has been changed using the `translate()` method
    - Note: Angles are in radians, not degrees. Convert using:  $\text{radians} = (\text{Math.PI}/180) * \text{degrees}$ .
    - API: [Mozilla Developer Network](#)



# HTML5 Canvas Element - CanvasRenderingContext2D Interface



- rectangle 1: rotated based on the canvas origin
- rectangle 2: rotated from the center of the rectangle itself with the help of `translate()` method.
- JSFiddle: <https://jsfiddle.net/liviucotfas/2yf241q1/>

# HTML5 Canvas Element - CanvasRenderingContext2D Interface

- Scaling
  - `scale(x, y)`
    - Scales the canvas units by x horizontally and by y vertically.
    - Both parameters are real numbers. Values that are smaller than 1.0 reduce the unit size and values above 1.0 increase the unit size. Values of 1.0 leave the units the same size.
    - Using negative numbers you can do axis mirroring
    - API: [Mozilla Developer Network](#)

# HTML5 Canvas Element - CanvasRenderingContext2D Interface



- rectangle: scaled
- text: mirrored.
- JSFiddle: <https://jsfiddle.net/liviucotfas/Lyycaq7gv/>

# HTML5 Canvas Element - CanvasRenderingContext2D Interface

- Combining Transforms
  - the browser is using a transformation matrix

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

- the translate, rotate, and scale methods end up affecting the values stored by this matrix

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

# HTML5 Canvas Element - CanvasRenderingContext2D Interface

- Transforms
  - allow modifications directly to the transformation matrix
  - `transform(m11, m12, m21, m22, dx, dy)`
    - multiplies the current transformation matrix with the matrix described by its arguments.
  - API: [Mozilla Developer Network](#)

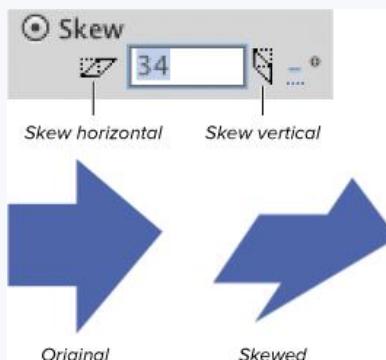
$$\begin{pmatrix} m_{11} & m_{21} & d_x \\ m_{12} & m_{22} & d_y \\ 0 & 0 & 1 \end{pmatrix}$$

# 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: coordinates transformation

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

- resetTransform()
  - resets the current transform to the identity matrix. This is the same as calling:  
`ctx.setTransform(1, 0, 0, 1, 0, 0);`

# HTML5 Canvas Element - CanvasRenderingContext2D Interface

- `transform(m11, m12, m21, m22, dx, dy)`
  - resets the current transform to the identity matrix, and then invokes the `transform()` method with the same arguments.
  - basically undoes the current transformation, then sets the specified transform, all in one step.

# HTML5 Canvas Element - ImageData Interface

- The ImageData interface represents the underlying pixel data of an area of a <canvas> element.
- API: [Mozilla Developer Network](#)

# HTML5 Canvas Element - ImageData Interface

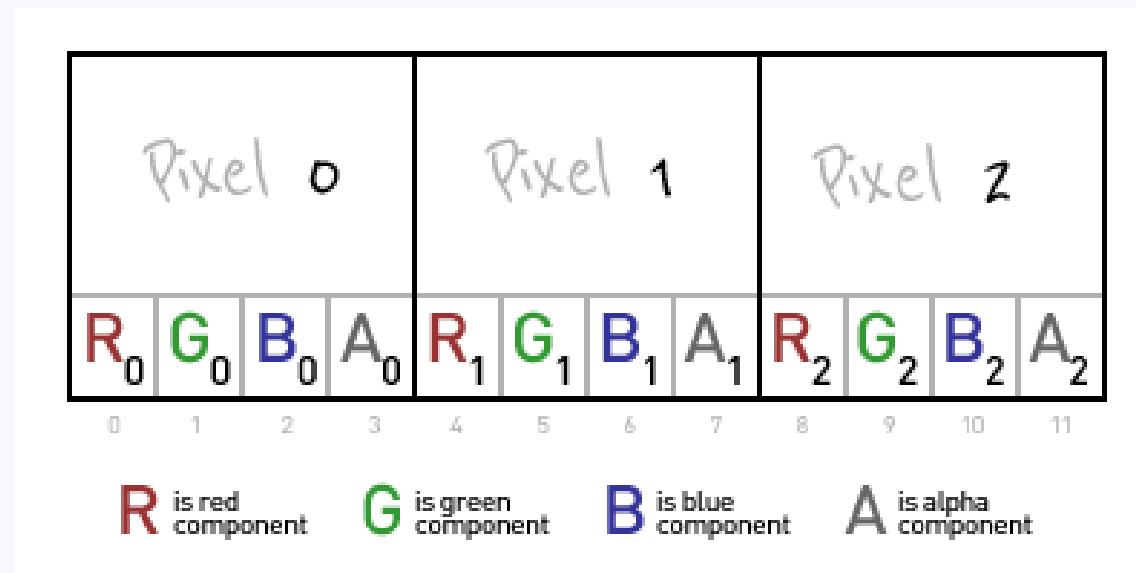
- It is created using the creator methods on the CanvasRenderingContext2D object associated with a canvas:
  - `CanvasRenderingContext2D.createImageData()`: creates a new, blank ImageData object with the specified dimensions. All of the pixels in the new object are transparent black.
  - `CanvasRenderingContext2D.getImageData(sx, sy, sw, sh)`: returns an ImageData object representing the underlying pixel data for the area of the canvas denoted by the rectangle which starts at (sx, sy) and has an sw width and sh height

# HTML5 Canvas Element - ImageData Interface

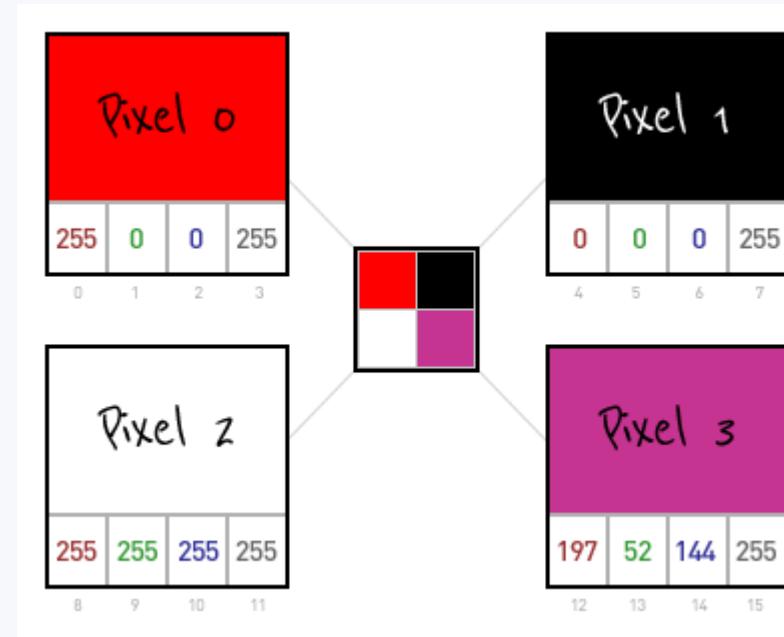
- It can also be used to set a part of the canvas by using:
  - void ctx.putImageData(imagedata, dx, dy): paints data from the given ImageData object onto the bitmap at the given coordinates.
- JSFiddle: <https://jsfiddle.net/liviucotfas/64fzcea/>

# HTML5 Canvas Element - ImageData Interface

- Properties
  - `ImageData.data` - a `Uint8ClampedArray` representing a one-dimensional array containing the data in the RGBA order, with integer values between 0 and 255 (included).



# HTML5 Canvas Element - ImageData Interface



```
[255,0,0,255, 0,0,0,255, 255,255,255,255, 203,53,148,255]
```

## HTML5 Canvas Element - ImageData Interface

- `ImageData.height` - an unsigned long representing the actual height, in pixels, of the `ImageData`.
- `ImageData.width` - an unsigned long representing the actual width, in pixels, of the `ImageData`.

# HTML5 Canvas Element - ImageData Interface

- iterating over all the pixels in a canvas

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

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

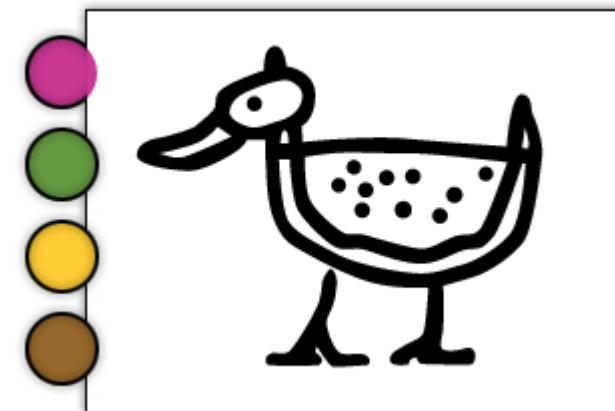
        var red = imageData.data[i]; // [0..255]
        var green = imageData.data[i+1]; // [0..255]
        var blue = imageData.data[i+2]; // [0..255]
        var transparency = imageData.data[i+3]; // [0..255]
    }
}
```

# 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](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#A color picker](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Pixel manipulation with canvas#A color picker)

# HTML5 Canvas Element

- Examples:
  - <http://www.williammalone.com/articles/html5-canvas-javascript-paint-bucket-tool/>
  - [www.williammalone.com/articles/create-html5-canvas-javascript-drawing-app/#demo-complete](http://www.williammalone.com/articles/create-html5-canvas-javascript-drawing-app/#demo-complete)



# Effects

- Color Filter

- Red filter:  $r' = r; g' = 0; b' = 0$

- Negative

- $r' = 255 - r; g' = 255 - g; b' = 255 - b;$

- Greyscale

- $r' = g' = b' = (r + g + b) / 3$
  - $r' = g' = b' = 0.299 * r + 0.587 * g + 0.114 * b$

# Effects

- Brightness
  - $r' = r + \text{value};$  if ( $r' > 255$ )  $r' = 255$  else if ( $r' < 0$ )  $r' = 0;$
  - $g' = g + \text{value};$  if ( $g' > 255$ )  $g' = 255$  else if ( $g' < 0$ )  $g' = 0;$
  - $b' = b + \text{value};$  if ( $b' > 255$ )  $b' = 255$  else if ( $b' < 0$ )  $b' = 0;$
- Threshold
  - $v = (0.2126*r + 0.7152*g + 0.0722*b) \geq \text{threshold} ? 255 : 0; r' = g' = b' = v$

# Effects

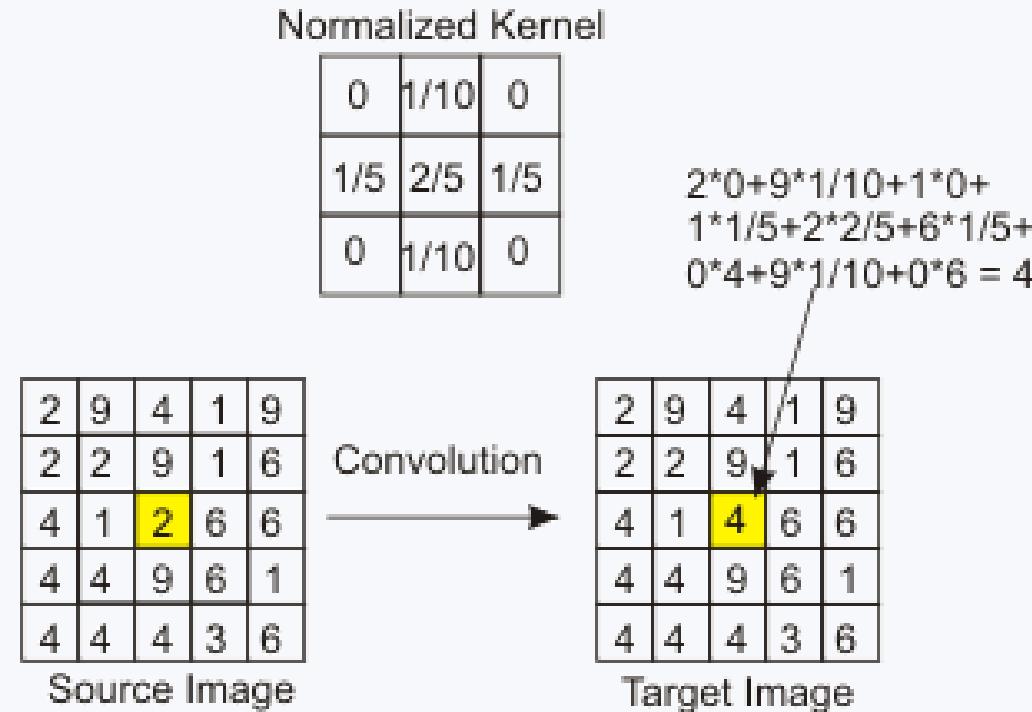
- Examples:
  - <https://www.html5rocks.com/en/tutorials/canvas/imagefilters/#toc-simplefilters>
- Further reading / examples:
  - [https://developer.mozilla.org/en-US/docs/Web/API/Canvas\\_API/Tutorial/Pixel manipulation with canvas](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Pixel manipulation with canvas)

# Effects - Convolution filters

- Allow us to implement generic filters for image processing: blurring, sharpening, embossing, edge detection.
- Calculates the new value for a pixel based on the values of nearby pixels in the original image.
- Example: blur filter 
$$\begin{bmatrix} 1/9, 1/9, 1/9, \\ 1/9, 1/9, 1/9, \\ 1/9, 1/9, 1/9 \end{bmatrix}$$
- Note: to maintain the brightness of the image, the sum of the matrix values should be one.

# Effects - Convolution filters

- Examples: <https://www.html5rocks.com/en/tutorials/canvas/imagefilters/#toc-convolution>



# WebGL

- Documentation: [https://developer.mozilla.org/en-US/docs/Web/API/WebGL\\_API](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/>

# Vector Graphics

# Vector Graphics

- In **bitmapped graphics**, the computer is given a detailed description of an image that it then matches, pixel by pixel.
- In **vector-drawn graphics**, the computer is given a **set of commands** that it executes to draw the image. Pictures contain *paths* made up of points, lines, curves and shapes.
- A **vector** is a line with a particular length, curvature, and direction. **Vector graphics** are composed of lines that are mathematically defined to form shapes such as rectangles, circles, triangles, and other polygons.

# Vector Graphics

- Each vector path forms the outline of a geometric region containing color information.

# Vector Graphics

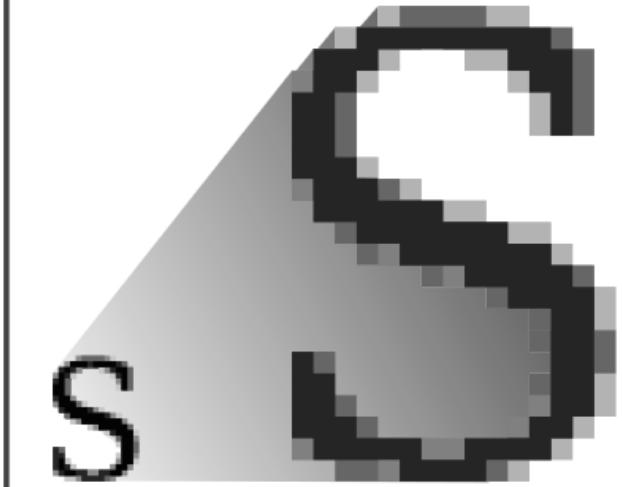


# Vector Graphics

- Because paths can be mathematically resized, vector graphics can be scaled up or down without losing any picture clarity.

# Scaling / Zoom

GIMP



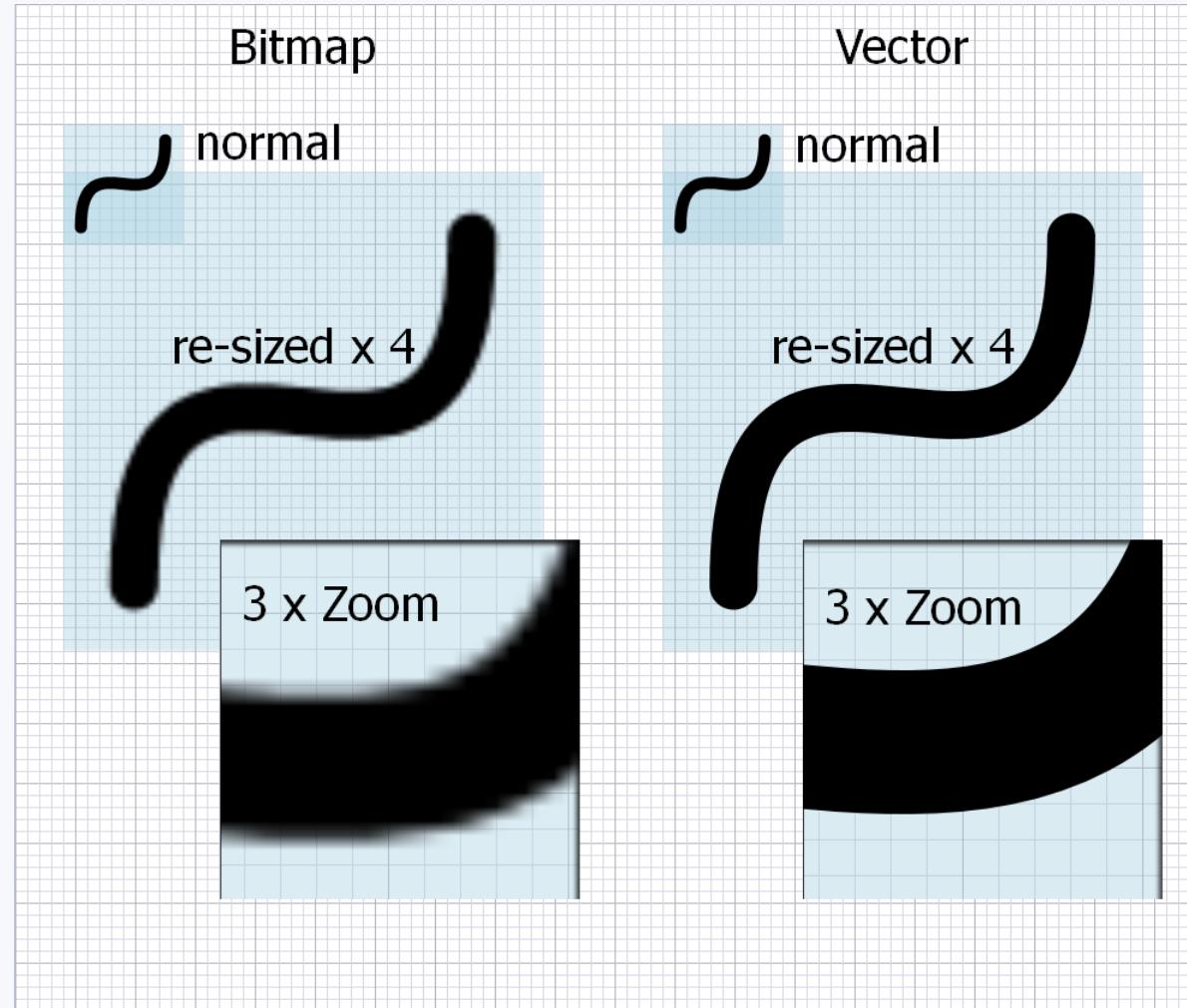
**BITMAP**  
.jpeg .gif .png

INKSCAPE



**OUTLINE**  
.svg

# Scaling / Zoom



# Animation

- You could also use vector graphics to create an animation, such as with Flash. Instead of drawing every separate frame of your project—with 24 frames appearing each second—you could create two different graphics for a segment and let your animation software mathematically interpolate the positions of the components in the in-between frames (a technique known as tweening).

# Advantages and Disadvantages

- Advantages:
  - For relatively simple images, the list of drawing commands takes up much less file space than a bitmapped version of the same graphic.
  - A draw program might use a command similar to "RECT 300, 300, RED" to create a red square with sides of 300 pixels. The file for this image contains 15 bytes that encode the alphanumeric information in the command.
  - The same image could also be created with a paint program as a bitmapped graphic. Using 8-bit color resolution (one byte per pixel), this file would require 90,000 bytes ( $300 \times 300$ ). The much smaller files sizes of drawn images can be a significant advantage for multimedia developers.

# Advantages and Disadvantages

- Another advantage of vector-drawn graphics is smooth **scaling**. Vector images are enlarged by changing the parameters of their component shapes. The new image can then be accurately redrawn at the larger size without the distortions typical of enlarged bitmapped graphics.
- Images that are needed in several sizes, such as a company logo, are often best handled as vector-drawn graphics.

# Advantages and Disadvantages

- Disadvantages:
  - The principal disadvantage of vector-drawn graphics is lack of control over the individual pixels of the image. As a result, draw programs cannot match the capabilities of bitmapped applications for display and editing of photo-realistic images.

# Formats

- SVG (Scalable Vector Graphics)
  - XML-based vector image format for two-dimensional graphics
  - provides supports for interactivity and animation
- DXF (Drawing Exchange Format)
  - is a CAD data file format developed by Autodesk for enabling data interoperability between AutoCAD and other programs.

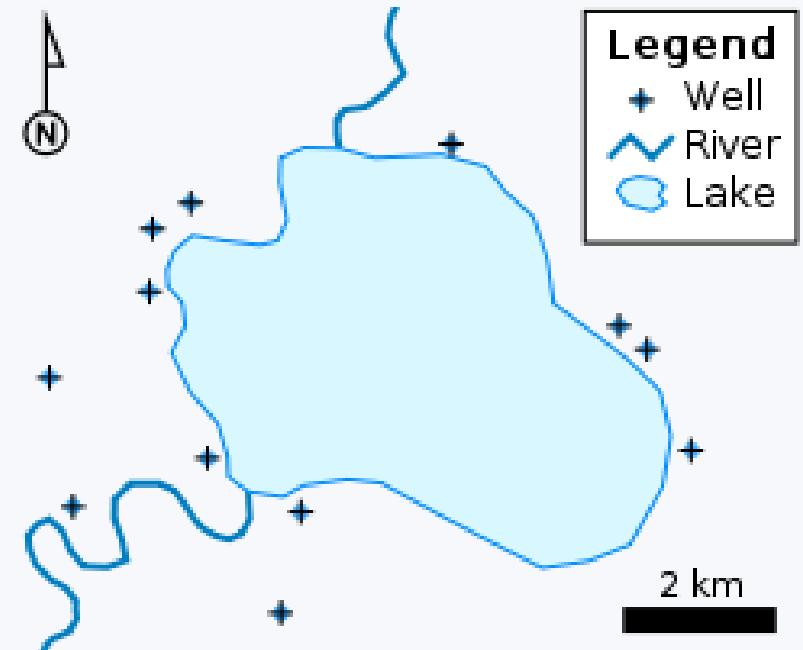
## Formats

- EPS (Encapsulated Post Script)
  - Created by Adobe Systems for representing vector graphics
  - Uses a computer language called Post Script
- SHP (Shapefile)
  - developed and regulated by Esri as a (mostly) open specification for data interoperability among Esri and other GIS software products
  - popular geospatial vector data format for geographic information system (GIS) software

# Vector Graphics Formats

Simple vector map that can be stored as Shape file:

- points (wells),
- polylines (rivers), and
- polygons (lake).

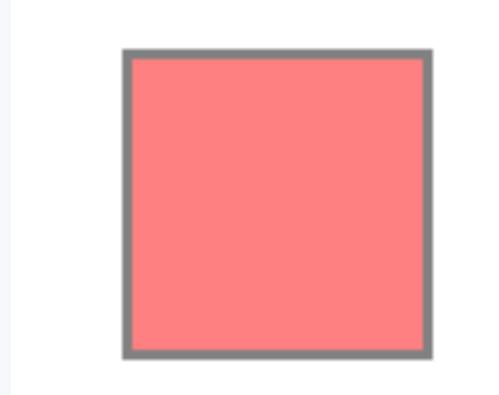


# SVG – Scalable Vector Graphics

- XML-based vector image format for two-dimensional graphics
- provides supports for interactivity and animation

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

# SVG – Scalable Vector Graphics



# SVG – Scalable Vector Graphics

- Creating SVG Images
- SVG images can be created with any text editor, but it is often more convenient to create SVG images with a drawing program, like [Inkscape](#).

# SVG – Scalable Vector Graphics

- Line

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

- Example:

- [http://www.w3schools.com/graphics/svg\\_line.asp](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

- Rectangle

```
<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](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

- Circle

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

- Example
  - [http://www.w3schools.com/graphics/svg\\_circle.asp](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

- Elipse

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

- Polygon

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

- Polyline

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

- Text

```
<text x="start-x" y="start-y">continut</text>
```

# SVG – Scalable Vector Graphics

- Grouping elements

```
<g id="id_grup">... <!-- elemente --> </g>
```

- Defining groups

```
<defs>... <!-- definire grupuri --> </defs>
```

- Reusing groups

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

## Vector Graphics

# SVG – Scalable Vector Graphics

- Interacting with SVG using CSS
- Link: [https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting started/SVG and CSS](https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started/SVG_and_CSS)

# SVG – Scalable Vector Graphics

- Interacting with SVG using JavaScript / jQuery
  - similar to the approach used for HTML elements;
- Particularities
  - when creating an element we need to use the SVG namespace

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

## Vector Graphics

# SVG – Scalable Vector Graphics

- Example

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

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

# Digital Video

## Definition

- Digital video is a representation of **moving visual images** in the form of encoded digital data. This is in contrast to analog video, which represents moving visual images with analog signals.
- Digital video comprises a series digital images displayed in rapid succession. In contrast, one of the key analog video methods, motion picture film, uses a series of photographs which are projected in rapid succession.
- Digital video was first introduced commercially in 1986 with the Sony D1 format, which recorded an uncompressed standard definition component video signal in digital form instead of the high-band analog forms that had been commonplace until then.

## Advantages

- Digital video can be processed using specialized computer software.
- Digital video can be **copied** with no degradation in quality. In contrast, when analog sources are copied, they experience generation loss.
- Digital video can also be **stored** on hard disks or **streamed** over **the Internet** to end users who watch content on a desktop computer screen or a digital Smart TV.
- The soundtrack is also stored as digital audio .

# Characteristics

- Screen Resolution
- Color Depth
- Frame Rate
- Interlaced / Progressive
- Compression Method (codec)

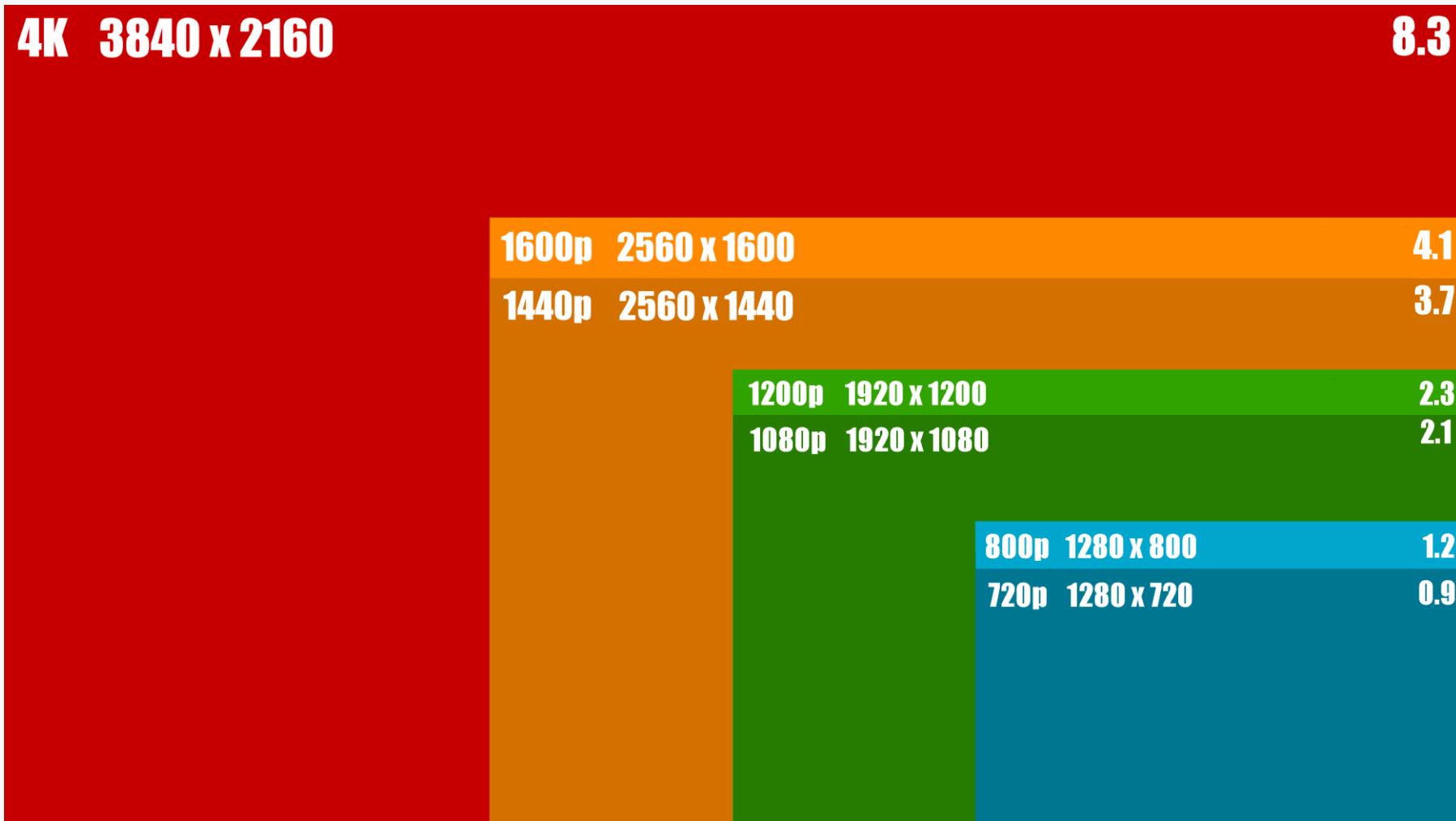
## Characteristics - Screen Resolution

- means the **width** and **height** of the displayed image, measured in pixels. In other words, the total number of pixels contained in each individual frame (**also called Spatial resolution**) [1]
- the smaller the screen resolution of a digital video, the less processing, storage, and transmission it requires [2].
- The output resolution of the relatively high-quality DV (digital video) format is **720 × 480 pixels**. This requires the computer to process and transmit information for nearly **350,000 pixels** at rates of up to 30 times per second [2].
- Reducing display size to **176 × 144** for mobile phones yields a pixel count of approximately **25,000** [2].

# Characteristics – Common Screen Resolution

Name	Pixels (width x height)	Aspect Ratio	Notes
<i>Standard Definition (SD)</i>			
<b>480p / 480i</b>	720x480 (or 704x480)	4:3 (approx)	NTSC
<b>576p / 576i</b>	720x576 (or 704x576)	4:3 (approx)	PAL
<i>High Definition (HDTV)</i>			
<b>720p</b>	1280x720	16:9	
<b>1080p / 1080i</b>	1920x1080	16:9	
<i>Ultra High Definition (UHDTV)</i>			
<b>4K (2160p)</b>	3840x2160	16:9	Exactly 4 × 1080p
<b>8K (4320p)</b>	7680x4320	16:9	Exactly 16 × 1080p
<b>8640p</b>	15360x8640	16:9	Exactly 32 × 1080p
<i>Digital Cinema (DCI)</i>			
<b>2K</b>	2048 × 1080	1.90:1	The first generation of digital cinema projectors.
<b>4K</b>	4096 × 2160	1.90:1	2nd generation digital cinema.

# Characteristics - Screen Resolution



# Characteristics - Screen Resolution



## Characteristics - Color Depth

- Color **depth** is the number of bits used to indicate the color of a single **pixel** in a video frame buffer, or the number of bits used for each color component of a single **pixel**
- Similar to raster images, the more bits are used for storing the color, the more subtle variations of colors can be reproduced.

## Characteristics – Frame Rate

- Frame rate, also known as frame frequency, is the frequency (rate) at which an imaging device displays consecutive images called frames.
- The perception of continuous motion in video is dependent on two factors. First, images must be displayed rapidly enough to allow persistence of vision to fill the interval between frames. Second, the changes in the location of objects from frame to frame must be relatively gradual.
- Lowering the frame rate both slows the delivery of individual images and drops out frames of video. If the rate is low enough, viewers will experience a string of still images with abrupt changes of content—in other words, a jerky video.

## Characteristics – Frame Rate

- A common frame rate for broadcast video is 30 frames per second (fps). Video intended for streaming over the Internet is often delivered at a rate of just 15 fps, effectively cutting the required data rate in half [1]
- NTSC used about 30 frames per second and PAL used 25 frames per second [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.

# Characteristics – Interlaced / Progressive

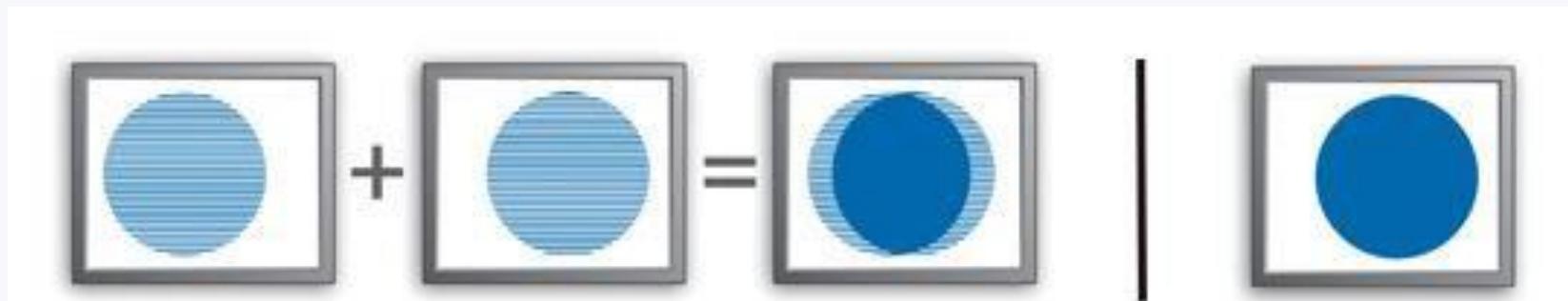


# Characteristics – Interlaced / Progressive

- 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.



## Characteristics – Compression

- **Redundancy** - the amount of wasted space consumed by storage media to record picture information in a digital image / digital video.
- The goal of **video compression** is two-fold:
  1. to reduce the file size of an image by eliminating or rewriting as much of the redundant information as possible;
  2. to retain the visible quality of an image.

# Characteristics – Compression

- Redundancies can occur:
  - within the two-dimensional space of a single frame of video (as with a photograph) - *spatial redundancy*.
  - across time in a video sequence containing many frames – *temporal redundancy*.
- Example: a five-second (150 frames) shot of a blue sky on a cloudless day.
  - thousands of blue pixels retain the same color value across the entire sequence of 150 frames. This phenomenon is called *temporal redundancy* and occurs whenever the value of a pixel remains unchanged from one frame to the next in a time-based sequence. The pixels also stretch outward within the space of each individual frame. This phenomenon is called *spatial redundancy*.

# Characteristics – Compression



Spatial redundancy

# Characteristics – Compression



Temporal redundancy (1 second / 30 Frames)

# Characteristics – Compression

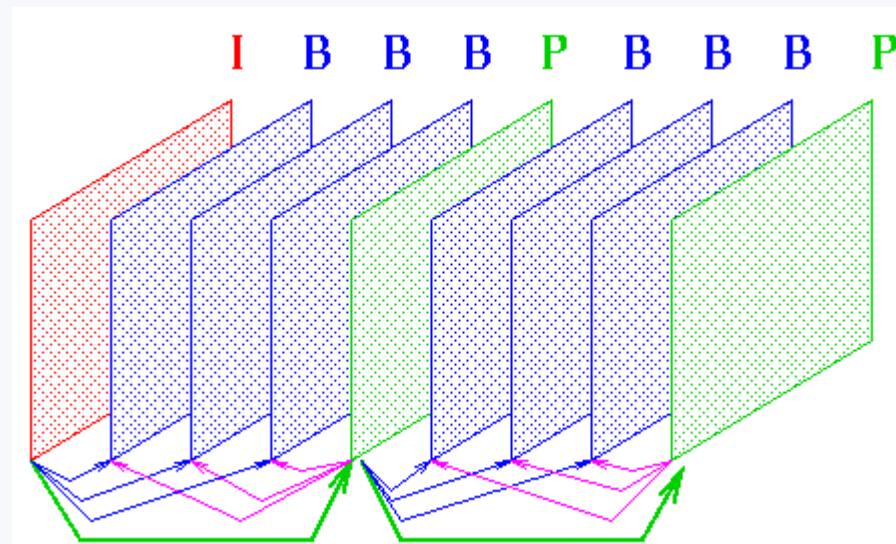
- *Intraframe (or I-frame) compression*
  - Eliminates spatial redundancies “within” a video frame in much the same way that JPEG compression is used to reduce them in a digital photograph.
  - I-frames are typically compressed at a ratio of 10:1. This means that a compressed I-frame consumes as little as 10% of the file space of a raw uncompressed frame.
  - Since I-frames are fully defined by information from within the frame, they can be easily decoded and rendered onscreen during playback and editing.
  - However, the overall amount of compression that’s achieved with this approach is limited since temporal redundancies are not addressed.

# Characteristics – Compression

- *Interframe compression* (more common method)
  - exploits both spatial and temporal redundancies.
  - using the previous method of **intraframe compression**, all of the frames in a video stream are turned into I-frames. Each one is *intracoded* to eliminate spatial redundancy. Thus, compression is applied evenly to all of the frames within a video stream.
  - with **interframe compression**, I-frames are created at fixed intervals (typically every 15 frames). An I-frame marks the beginning of a packaged sequence of adjacent frames called a GOP (Group of Pictures). The fully defined I-frame serves as a *keyframe* or reference for other frames in the group. Its job is to hold repeating color values in place that will not change across the sequence. Basically, it creates a digital marker on the frame that says “do not change the color of this pixel until you are told to do so.”

# Characteristics – Compression

- MPEG Compression
  - exploits both spatial and temporal redundancies (interframe compression )
  - each I-frame is followed by a sequence of 14 frames designated as either a *P-frame* or a *B-frame*.



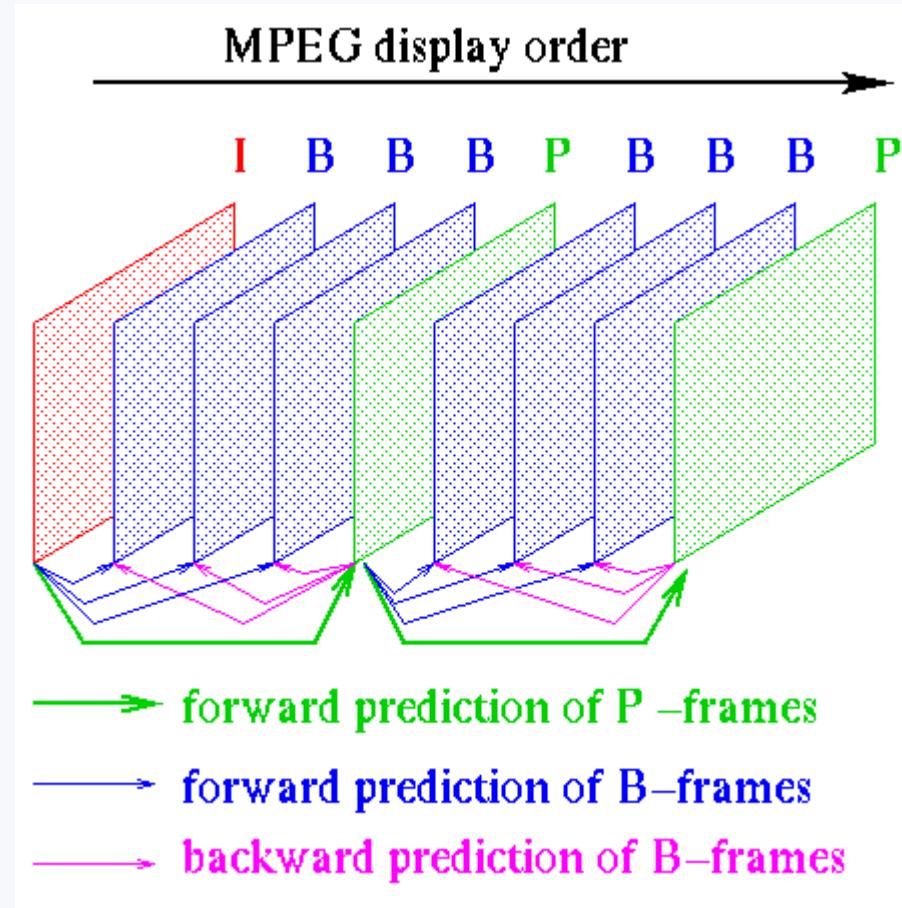
# Characteristics – Compression

- A *P-frame*
  - a predictive coded image that only stores data for pixels that are different from the preceding frame.
  - Example: in a shot of a bird flying across a blue sky, only pixels related to the moving bird would be encoded to a P-frame. The unchanged background pixels simply carry forward from information stored in the previous frame.
  - on average, a P-frame can be compressed twice as much as an I-frame.

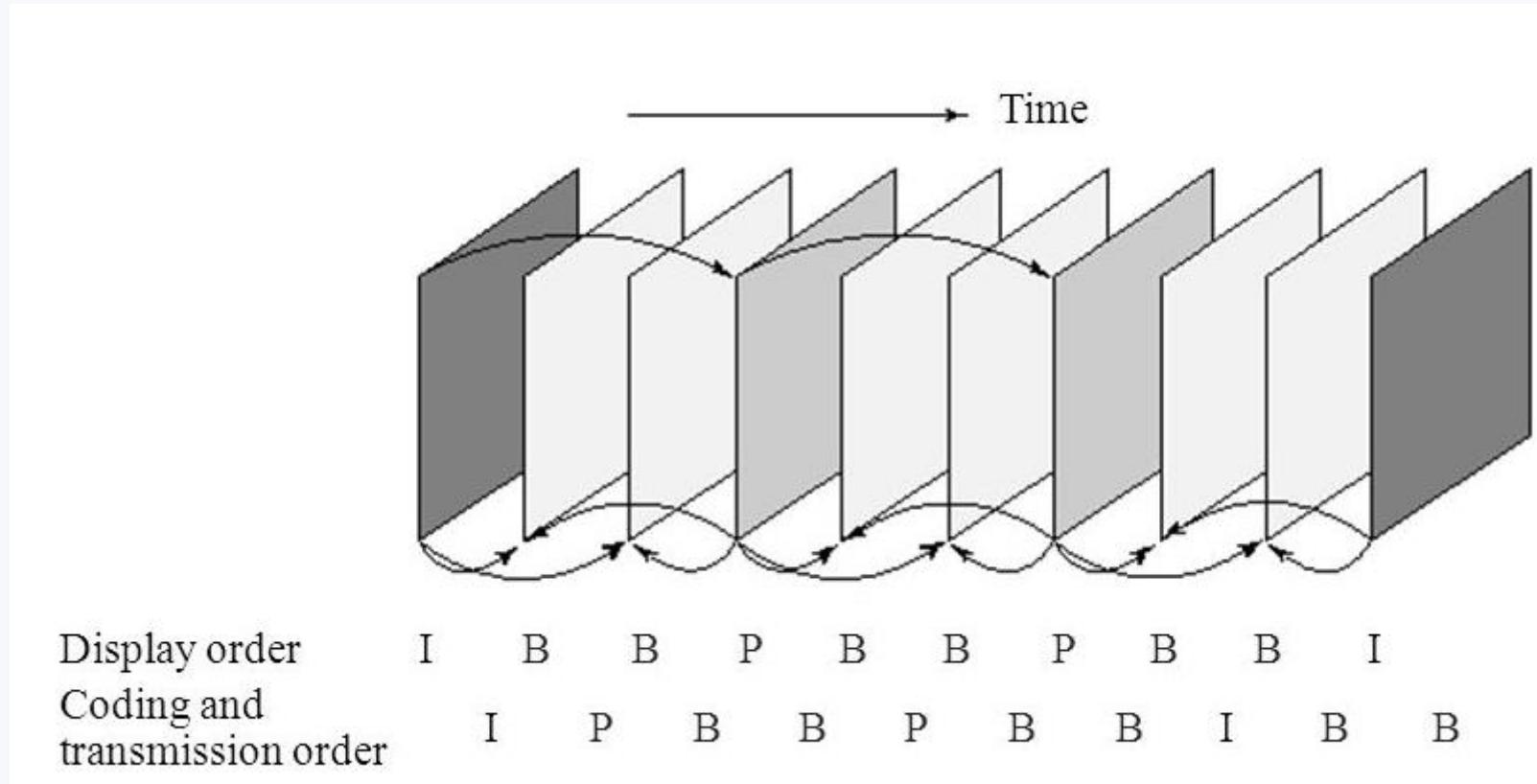
# Characteristics – Compression

- A *B-frame*
  - a bidirectional predictive coded image. It records changes from both the preceding frame and the one immediately following.
  - On average, a B-frame can be compressed twice as much as a P-frame.
- With **interframe encoding**, all frames are not equally compressed. The more heavily compressed P-frames and B-frames require more processing power to encode and decode.

# Characteristics – Compression



# Characteristics – Compression



# Characteristics – Video Formats - Containers

- Container:
  - exists solely for the purpose of bundling all of the audio, video, and codec files into one organized package.
  - in addition, the container often contains chapter information for DVD or Blu-ray movies, metadata, subtitles, and/or additional audio files such as different spoken languages.

# Characteristics – Video Formats - Containers

- **Matroska Multimedia Container MKV:**
  - file extension: .mkv .mk3d .mka .mks
  - Google \*.webm is based on the specifications of this format
  - open standard, free container format
  - supports almost any audio or video format which makes it adaptable, efficient, and highly regarded as one of the best ways to store audio and video files.
  - supports multiple audio, video and subtitle files even if they are encoded in different formats.
  - due to the options the container offers, as well as its handling of error recovery (which allows you to play back corrupted files), it has quickly become one of the best containers currently available.

## Characteristics – Video Formats - Containers

- **MPEG-4 (MP4):**
  - file extension: .mp4
  - digital multimedia container format most commonly used to store video and audio
  - recommended format for uploading video to the web
  - utilizes MPEG-4 encoding, or H.264, as well as AAC or AC3 for audio. It's widely supported on most consumer devices, and the most common container used for online video

# Characteristics – Video Formats - Containers

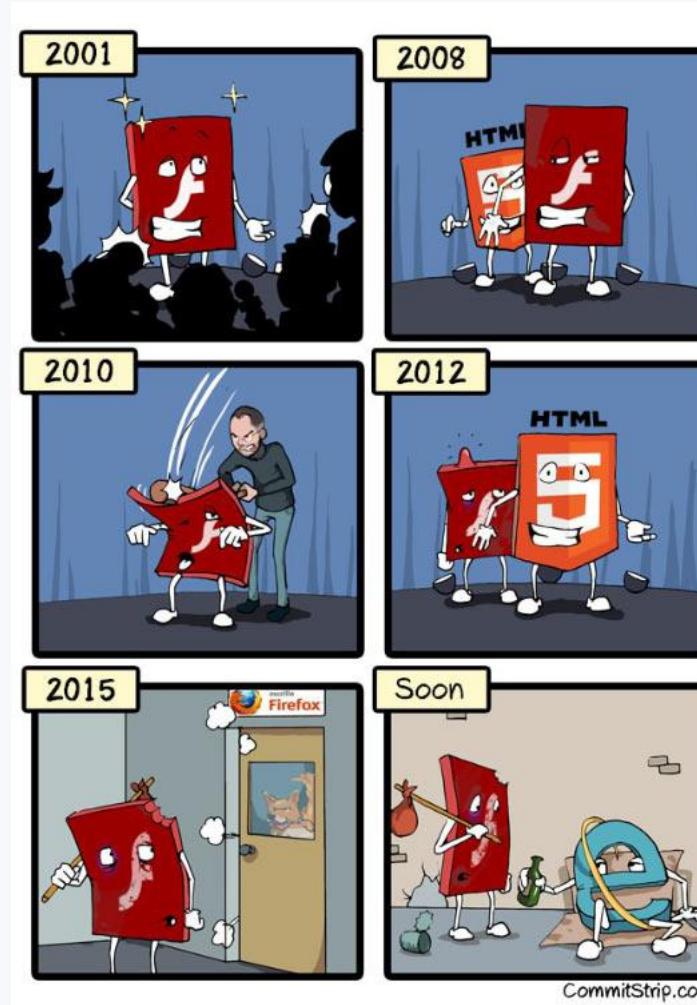
- Other popular Video Containers:
  - Ogg - free, open container format
  - AVI – Windows Professional
  - MOV – Mac everything
  - MPEG or MPG – by the MPEG group
  - FLV – Flash video
  - MP4 – by the MPEG group
  - VOB – DVDs
- Comparison:
  - [https://en.wikipedia.org/wiki/Comparison\\_of\\_video\\_container\\_formats](https://en.wikipedia.org/wiki/Comparison_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> - HTML element used to embed video content in a document. The video element contains one or more video sources. To specify a video source, use either the src attribute or the <source> element; the browser will choose the most suitable one.

```
<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>
  - attribute: *autoplay, controls, src, volume, ...*
  - API: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video>
- HTMLVideoElement (JavaScript)
  - properties: 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](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.clientWidth;
var H = canvas.height = video.clientHeight;

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

for (var y = 0; y < H; y++) {
    for (var x = 0; x < W; x++) {
        var i = (y * W * 4) + x * 4;
            // 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

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

    var video = $("#myVideo");

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

        video[0].src = lista[index];
        video[0].load();
        video[0].play();
    });
});
```

# Lab Examples

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



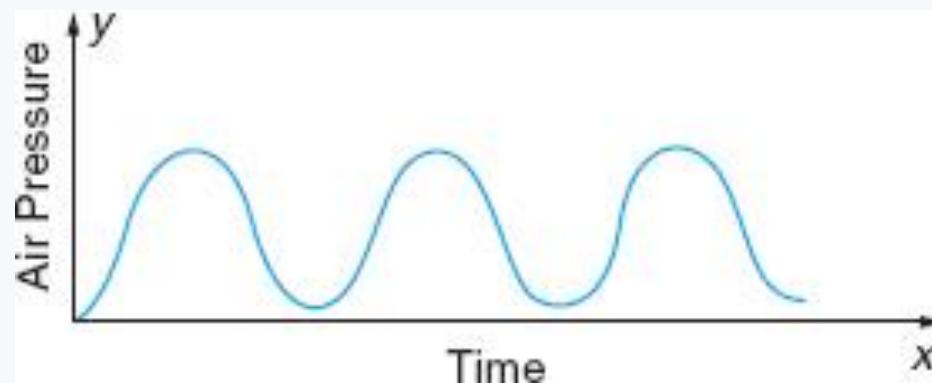
# Audio

# Sound

- Sound is a form of mechanical energy transmitted as vibrations in a medium.
- The medium is usually air, though sound can also be transmitted through solids and liquids.
- Example: a clap of the hands produces sound by suddenly compressing and displacing air molecules. The disturbance is transmitted to adjacent molecules and propagated through space in the form of a wave. We hear the hand clap when these vibrations cause motions in the various parts of our ears.

# Sound

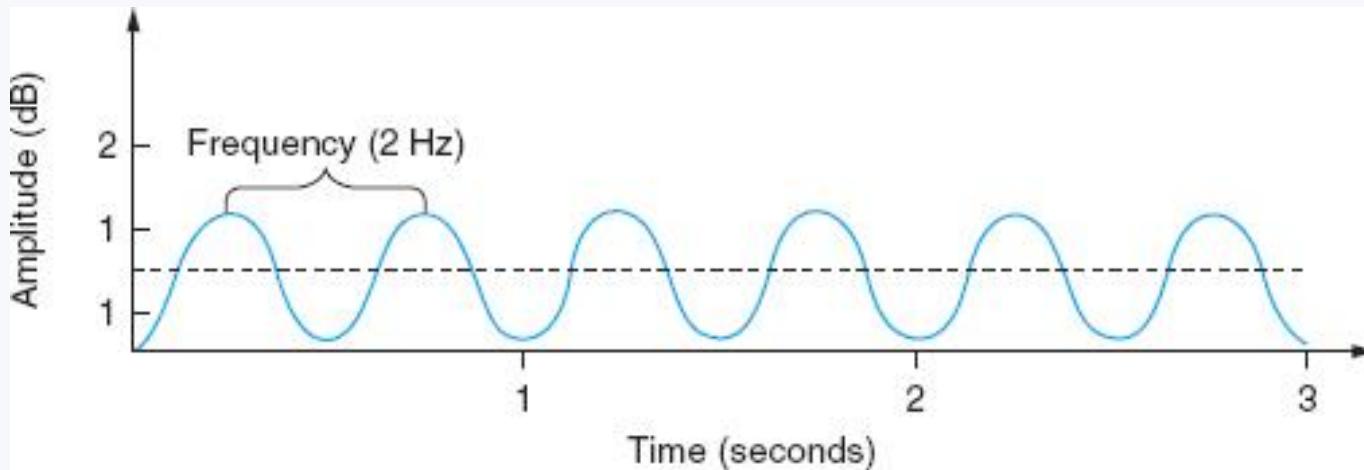
- Sound waves are often compared to the ripples produced when a stone is thrown into a pond. The stone displaces the water to produce high points, or wave peaks, and low points, or troughs. We see these alternating peaks and troughs as patterns of waves moving outward from the stone's point of impact. The clap of the hand also produces peaks and troughs as high air pressure is followed by a return to lower pressure. These changes in air pressure produce patterns of waves spreading in all directions from the sound's source.



# Sound

- Humans can hear sound waves with frequencies between about 20 Hz and 20 kHz
- **Noise** is unwanted sound judged to be unpleasant, loud or disruptive to hearing. From a physics standpoint, noise is indistinguishable from sound, as both are vibrations through a medium, such as air or water. The difference arises when the brain receives and perceives a sound.

# Representation

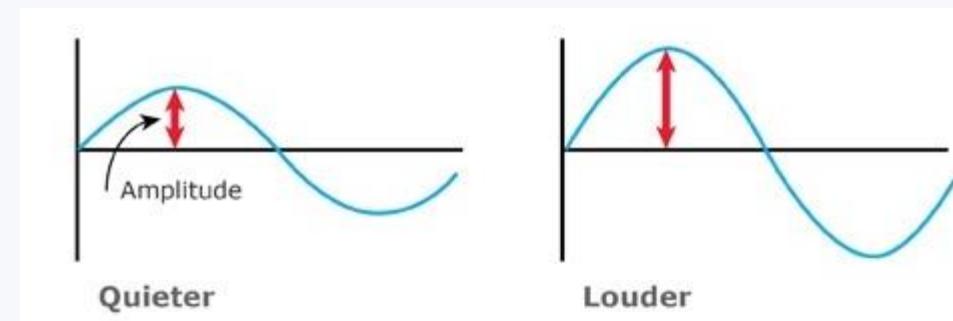


# Characteristics

- amplitude
- frequency
- duration

# Characteristics

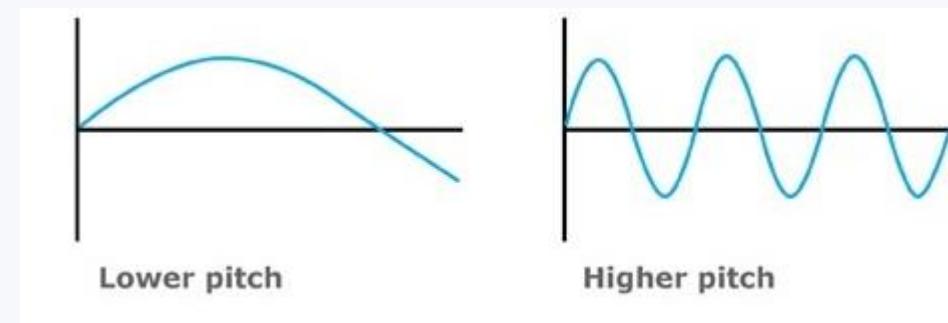
- **Amplitude** is a measure of sound pressure or the amount of energy associated with the sound. This is represented by the vertical, or *y*-axis, of the sine wave. Amplitude is perceived as the sound's **volume**, which is usually measured in **decibels (dB)**.



- In general, sounds with higher amplitudes are experienced as *louder*. The range of human hearing is approximately 3 to 140 dB. Each 10 dB increase roughly doubles the perceived volume of a sound.

# Characteristics

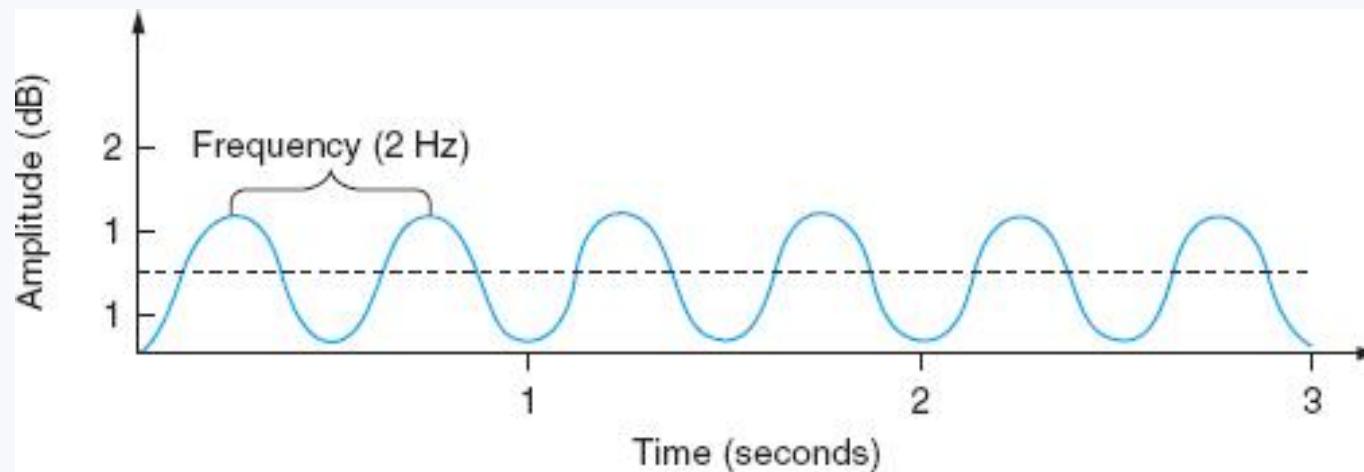
- **Frequency** is the number of times a waveform repeats in a given interval of time. It is represented on the horizontal axis as the distance between two wave peaks or troughs. Frequency is measured in *hertz* (Hz).
- One **hertz** is one repetition of a waveform in one second of time.



- Frequency is perceived as **pitch**. High frequencies produce sounds of higher pitch, and low frequencies produce low pitch. Pitch is the psychological perception of sound frequency. Humans can perceive a frequency range of 20 Hz to 20,000 Hz (or 20 *kilohertz* (kHz), *thousands* of hertz), though most adults cannot hear frequencies above 16 kHz.

# Characteristics

- The **duration** of the sound is the length of time it lasts. The total length of the horizontal axis represents duration

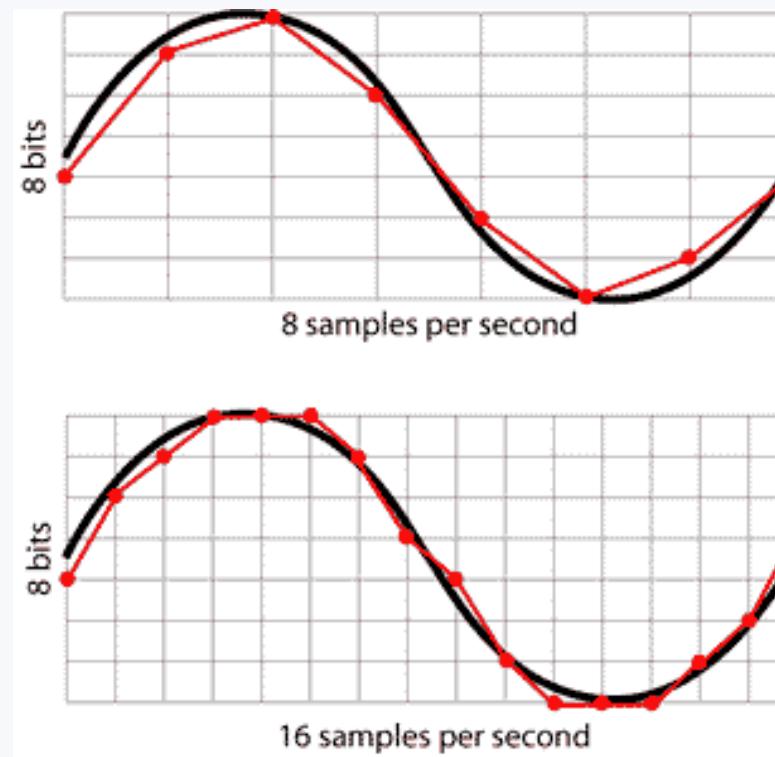


# Digital Audio

- Digital techniques represent sound as discrete (or discontinuous) elements of information.
- There are two major types of digital sound:
  - *sampled*
  - *synthesized*.

# Digital Audio

- Sampled sound is a digital recording of previously existing analog sound waves. A file for sampled sound contains many thousands of numerical values, each of which is a record of the amplitude of the sound wave at a particular instant, a *sampling* of the sound.



# Digital Audio

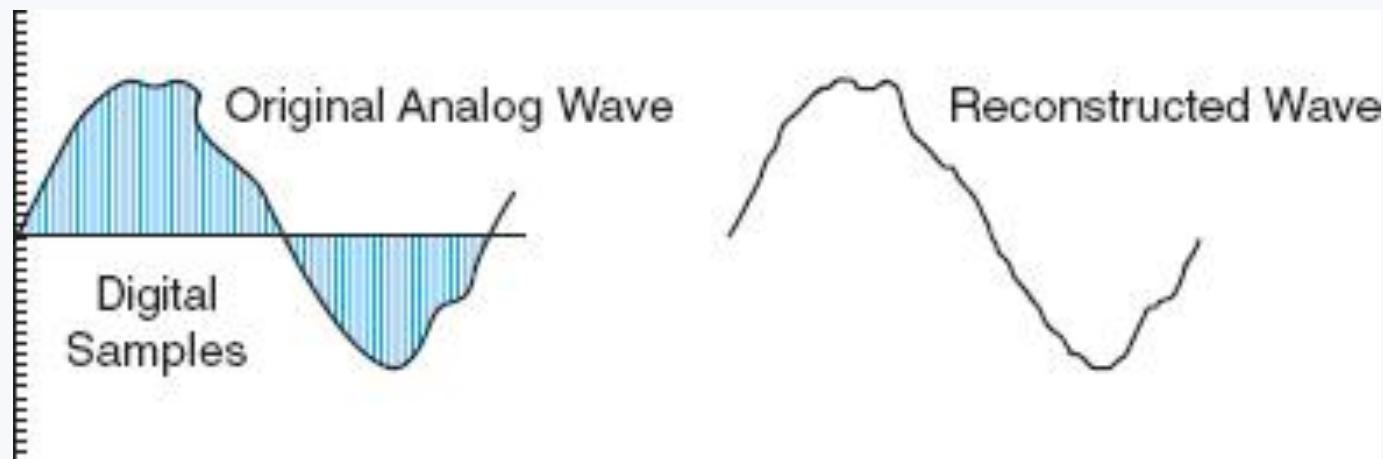
- **Synthesized** sound is new sound generated (or synthesized, “put together”) by the computer. A file for synthesized sound contains instructions that the computer uses to produce its own sound.
- **Sampling** is usually used to capture and edit naturally occurring sounds such as human speech, musical and dramatic performances, bird calls, rocket launches, and so on. **Synthesized** sound is generally used to create original musical compositions or to produce novel sound effects.

# Digital sampling

- capturing and storing sound in a digital format.
- the sound is captured by recording many separate measurements of the amplitude of a wave using an **ADC**, or *analog-to-digital converter*. An analog device, such as a microphone or the amplifier in a speaker system, generates a continuously varying voltage pattern to match the original sound wave. The ADC samples these voltages thousands of times each second. The samples are recorded as digital numbers. These digital values are then used to re-create the original sound by converting the digital information back to an analog form using a **DAC**, or *digital-to-analog converter*. The DAC uses the amplitude values to generate matching voltages that power speakers to reproduce the sound.

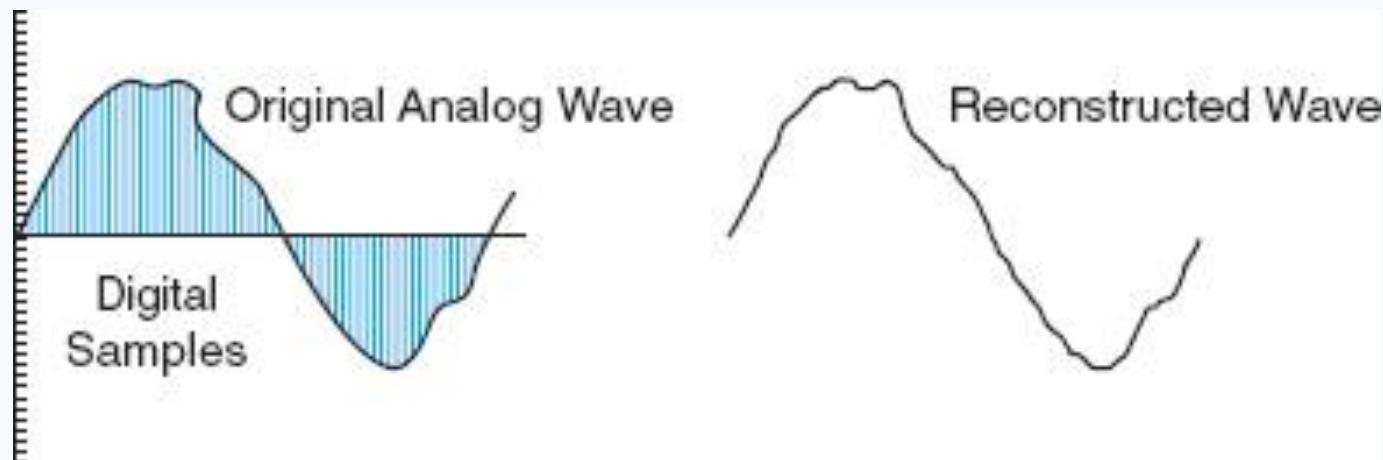
# Digital sampling

- Digital sampling replaces the continuous waveform of the original sound with a new wave created from a fixed number of discrete samples.
- Some information is always lost in sampling, because a continuous wave is infinitely divisible and sampling always yields a finite number of values.
- The quality of sampled sound is dependent on two factors directly connected to this sampling process: **sample resolution** and **sample rate**.



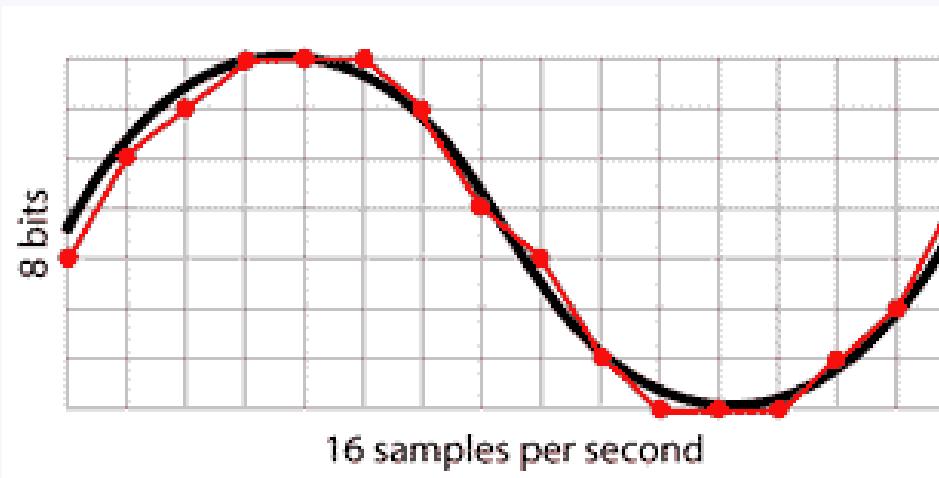
# Digital sampling

- Digital sampling replaces the continuous waveform of the original sound with a new wave created from a fixed number of discrete samples.
- Some information is always lost in sampling, because a continuous wave is infinitely divisible and sampling always yields a finite number of values.
- The quality of sampled sound is dependent on two factors directly connected to this sampling process: **sample resolution** and **sample rate**.



# Digital sampling

- **Sample Resolution** Each measurement of amplitude made by an ADC is recorded using a fixed number of bits. The number of bits used to encode amplitude is known as **sample resolution**.
- Sample resolutions for digital audio range from 8 to 32 bits, with the most common being the 16-bit CD-Audio standards and 24-bit DVD-Audio standards.



## Digital sampling

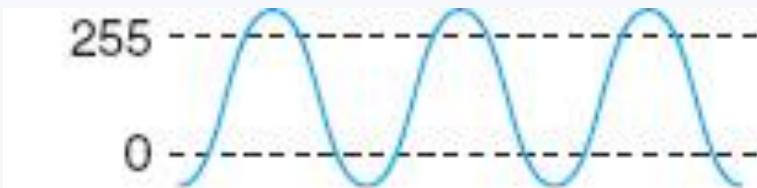
- Eight bits can record 256 different amplitude levels. This is adequate to capture the variations in limited decibel ranges, such as those between a human whisper and a shout, but higher sample resolutions are needed to accurately reproduce sounds with a wider range of amplitudes, such as musical performances. Thus, 8-bit audio is generally used only for simple sounds or in multimedia applications requiring very small file sizes.
- The CD-Audio standard, with its 16 bits per sample, supports over 65 thousand different amplitude levels, whereas the 24-bit DVD-Audio standard can represent over 16 million levels.

# Digital sampling

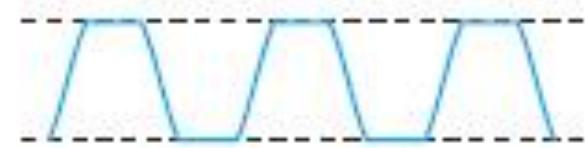
- Inadequate sample resolution can distort sound in two different ways: **quantization** and **clipping**.
- **Quantization** - each amplitude sample must be assigned one of the numbers available in the code being used. If the number of distinct values is too small, perceptually different amplitudes 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).

# Digital sampling

- **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.

# Digital sampling

- 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.

# Digital sampling

- 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.

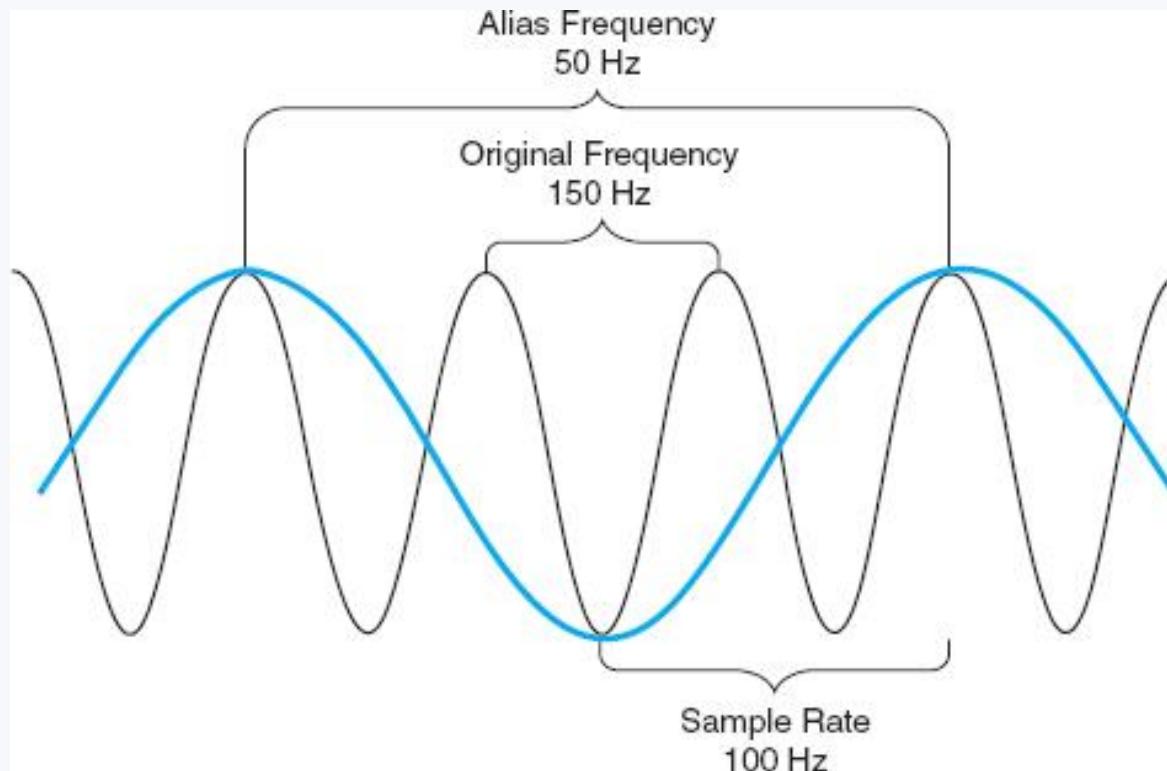
# Digital sampling

- **Sample rate** is the number of samples taken in a fixed interval of time. A rate of one sample per second is designated as a Hertz.
- Because sound samples are always taken thousands of times each second, sample rates are usually stated in kilohertz.
- Sample rate affects sound quality by determining the range of frequencies that can be represented in a digital recording. At least two measurements are required to capture each cycle of a sound wave—one for each high value, or peak, and one for each low value, or trough. The highest frequency that can be captured is thus one-half of the sample rate.
- CD-quality sound captures 44,100 samples per second (44.1 kHz sample rate) and can represent frequencies as high as 22,050 Hz or 22.05 kHz. DVD-Audio uses a sampling frequency of 96 kHz to capture frequencies as high as 48 kHz.

## Digital sampling

- Sounds that do not contain high frequencies can be more efficiently represented using lower sample rates because this will produce a smaller overall file size. One potential problem with lower sample rates, however, is **aliasing**.

# Digital sampling



# 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

# Sound Compression

- Lowering the sample rate and reducing sample resolution are two ways to reduce the size of a sampled sound file. These methods work well for sounds at relatively low frequencies and narrow amplitude ranges. They are not effective for sounds that contain wider ranges of both frequency and amplitude, such as musical performances. In these cases another strategy can be used: compression.
- Compression can be either **lossless** or **lossy**. Lossless compression uses more efficient coding to reduce the size of a file while preserving all the information of the original. Lossy compression discards some of the original information.

# Sound Compression

- Because **lossy strategies** produce much smaller files, they are the preferred technique for sound compression.
- Lossy sound compression **codecs** (coder/decoders) use various techniques to reduce file sizes. Some of these take advantage of **psychoacoustics**, the interplay between the psychological conditions of human perception and the properties of sound. For instance, while humans with optimal hearing can perceive frequencies as high as about 20 kHz, most people cannot distinguish frequencies above approximately 16 kHz. This means that higher-frequency information can be eliminated and most listeners will not miss it. Higher amplitude sounds in one stereo channel will also typically “drown out” softer sounds in the other channel. Again, this is information that usually will not be missed.

# Sound Compression

- Lossy compression also uses other techniques such as **variable bit rate encoding (VBR)**. In VBR, sounds are encoded using a different number of bits per second depending on the complexity of the sound. For simple passages of sound with limited frequencies, a smaller number of bits per second is used than for more complex passages, such as those with many different instruments and higher frequencies.
- Lossy codecs such as the widely supported **MP3** can reduce file sizes by as much as 80% while remaining virtually indistinguishable from the original CD-quality sound.

## Sampled Sound File Formats

- MP3 (MPEG1, audio layer 3)
  - extensions: .mp3 or .mpga
  - popular audio format that supports significant compression while preserving excellent quality.
  - widely supported across different computer platforms and is frequently used on the Internet.

# Sampled Sound File Formats

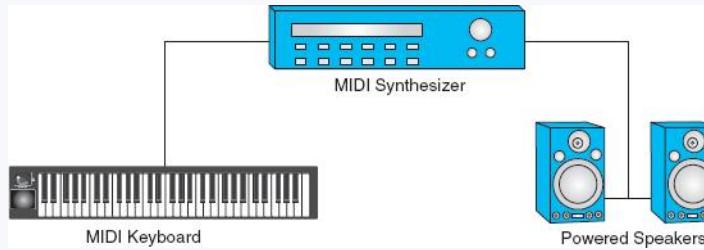
- AAC (advanced audio coding)
  - extension: .mp4 and .aac
  - the successor to MP3 specified in the MPEG4 standard, that produces better sound quality than MP3 at comparable bit rates.
  - it can also significantly reduce file sizes for comparable-quality audio.
  - many commercial users (including Apple iPod, Apple iPad, Apple iPhone, Blackberry, YouTube, and Sony PlayStation) have adopted the AAC standard for their digital audio.

# Sampled Sound File Formats

- Other sound file formats:
  - WMA (Windows Media Audio);
  - WAV;
  - ReadAudio;
  - AIFF;
  - AU.

## Audio

# Synthesized Sound



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

# Web Audio

- <audio> - HTML element is used to embed sound content in documents. It may contain one or more audio sources, represented using the src attribute or the <source> element; the browser will choose the most suitable one.

```
<audio controls>
  <source src="foo.wav" type="audio/wav">
  Your browser does not support the <code>audio</code>
  element.
</audio>
```

# Web Audio

- <audio>
  - attributes:
    - *autoplay*
    - *controls*
    - *loop*
    - *src*
    - *volume* (numeric) – value between 0 and 1
  - API: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/audio>

# Web Audio

- `HTMLAudioElement` (JavaScript)
  - provides access to the properties of `<audio>` elements, as well as methods to manipulate them. It derives from the `HTMLMediaElement` interface.
  - properties:
    - `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/HTMLAudioElement>

## Lab Example:

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



# Web Audio API

- The Web Audio API provides a powerful and versatile system for controlling audio on the Web, allowing developers to choose audio sources, add effects to audio, create audio visualizations, apply spatial effects (such as panning) and much more.
- API: [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Audio\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API)
- Web RTC API: <https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia>

# Web Audio

- Sound visualization using HTML5 Canvas: <http://nipe-systems.de/webapps/html5-web-audio/>
- Creating visualizations: [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Audio\\_API/Visualizations\\_with\\_Web\\_Audio\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Visualizations_with_Web_Audio_API)
- Record and save as MP3: <http://audior.ec/blog/recording-mp3-using-only-html5-and-javascript-recordmp3-js/> ,  
<https://www.html5rocks.com/en/tutorials/getusermedia/intro/>

# WebRTC API

- WebRTC (Web Real-Time Communications) is a technology which enables Web applications and sites to capture and optionally stream audio and/or video media, as well as to exchange arbitrary data between browsers without requiring an intermediary.
- The set of standards that comprises WebRTC makes it possible to share data and perform teleconferencing peer-to-peer, without requiring that the user install plug-ins or any other third-party software.
- Web RTC API: [https://developer.mozilla.org/en-US/docs/Web/API/WebRTC\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API),  
<https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia>

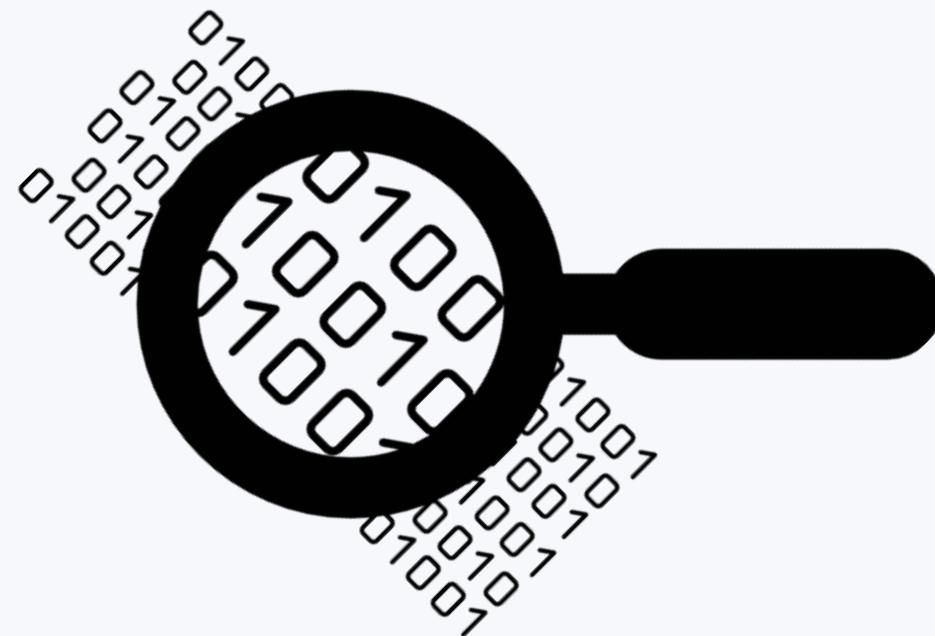
# Demo

- [https://developer.mozilla.org/en-US/docs/Web/API/WebRTC\\_API/Taking\\_still\\_photos](https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Taking_still_photos)



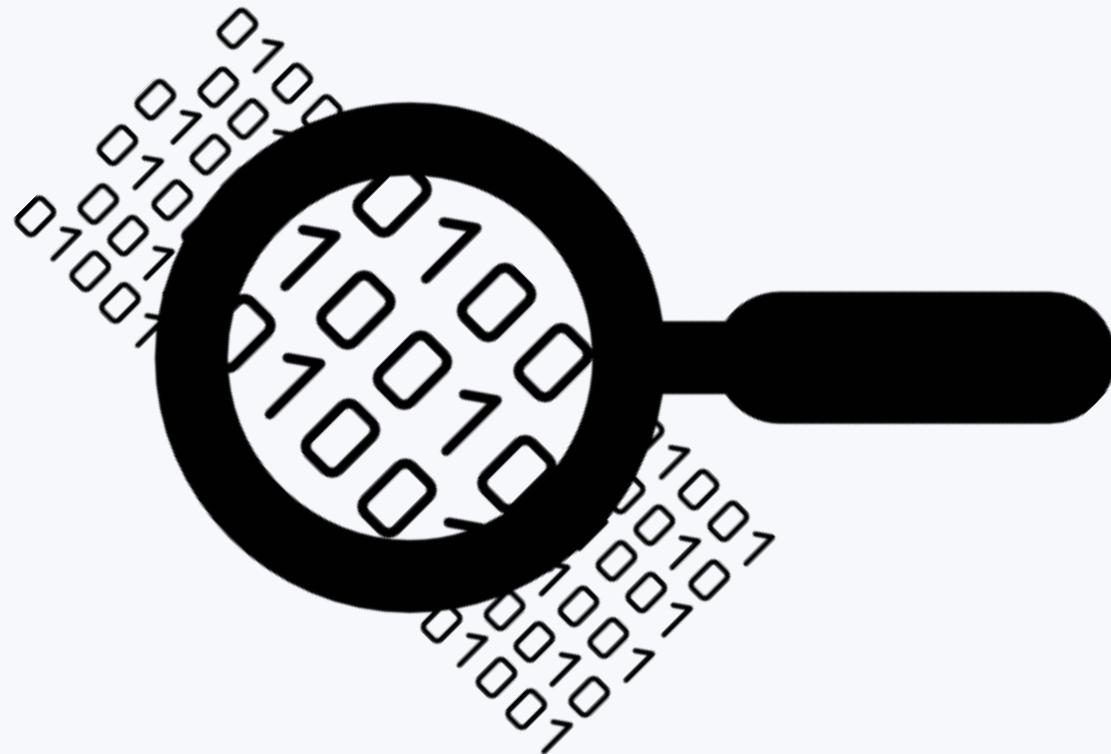
# Lab Example: Web Audio API

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



# Lab Example: Speech API

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



# Recommendations

- Consider the playback environment:
  - Public vs. private use (ex: autoplay)
  - Give users control:
    - Over volume.
    - To stop or start play.
- Avoid excessive use of sound. Sound can be more tiring for users than images or text

# Animation

# Animation

- 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

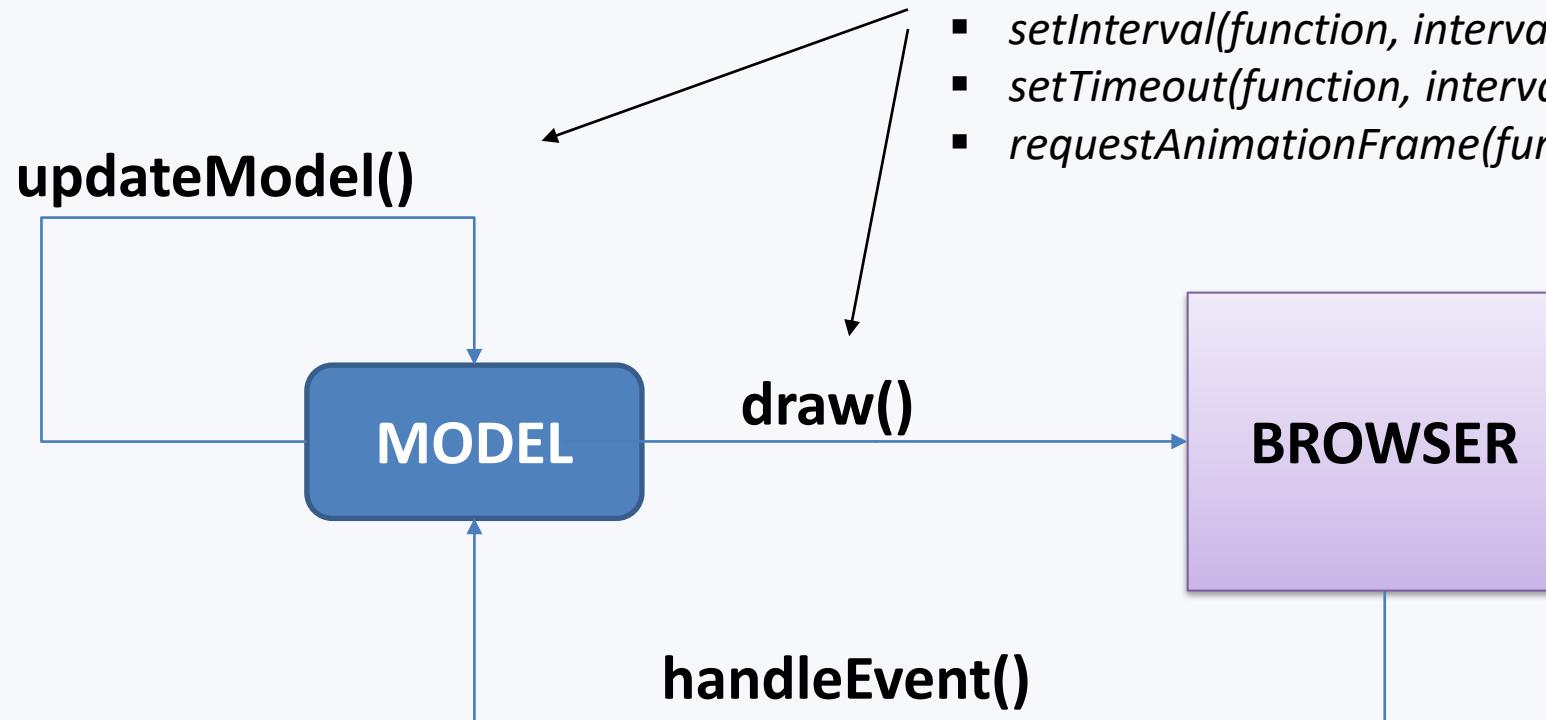
# Digital Animation

- 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

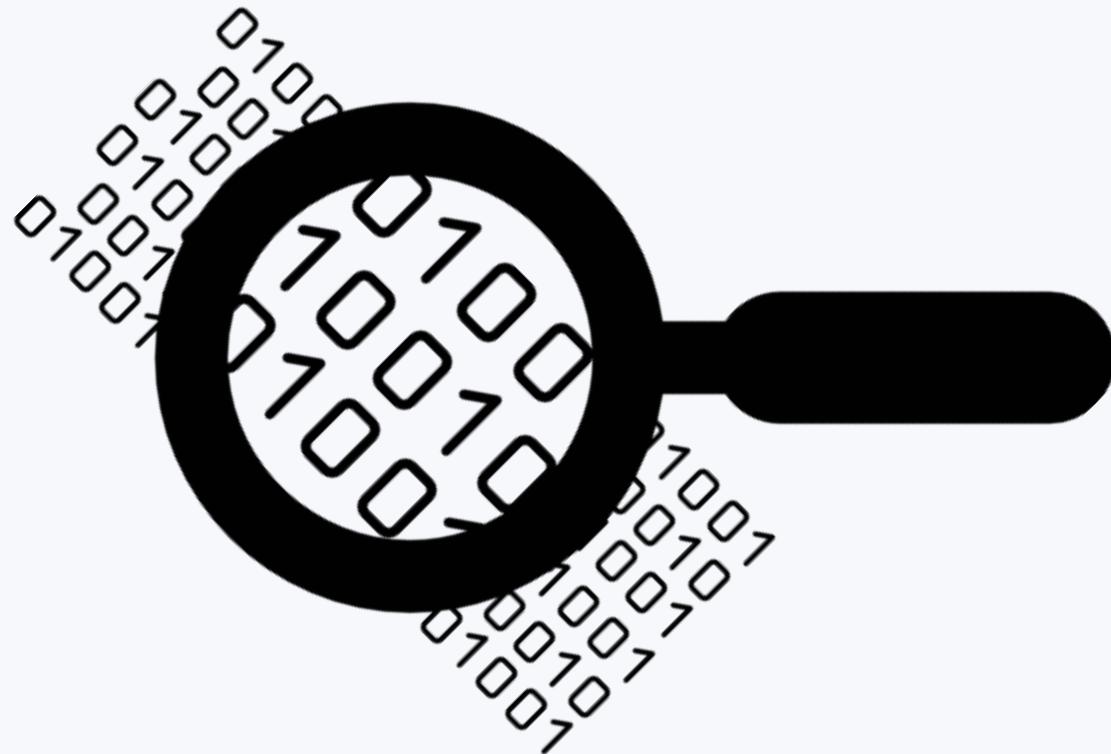
- size tweening - the first key frame is the object at its initial size and the second is the final size.
- alpha tweening- color and transparency can be animated using key frames representing initial and final image properties.
- 3. programmed animation

# Web Animation



# Demo

- lab animation examples



# CSS Transitions

- Guide: [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_Transitions/Using\\_CSS\\_transitions](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](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](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](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

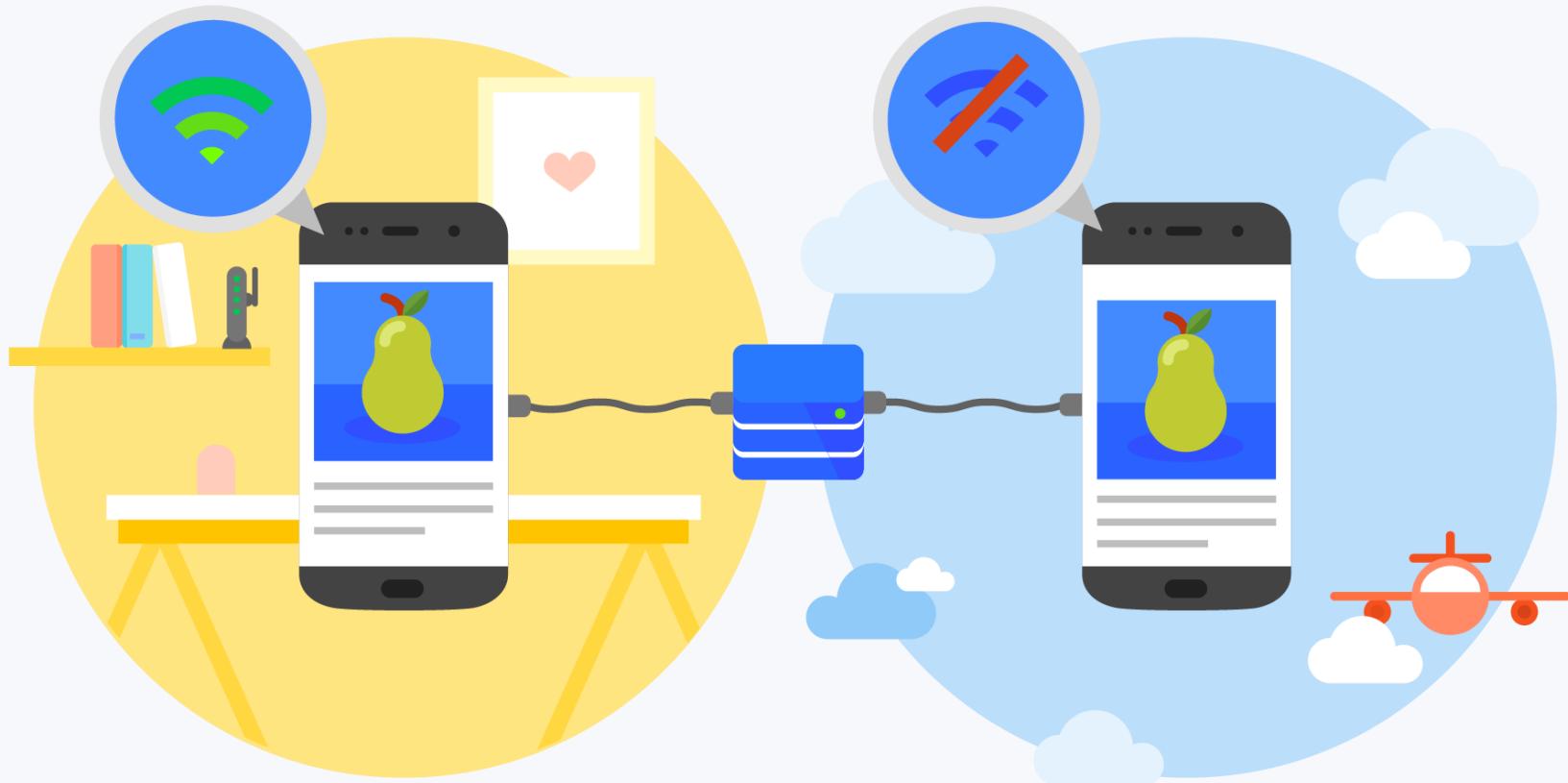
- An application developed using standard web technologies (HTML5, JavaScript, CSS3) which is:
- **Reliable** - load instantly even in uncertain network conditions.
- **Fast** - respond quickly to user interactions with silky smooth animations and no janky scrolling.
- **Engaging** - feel like a natural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

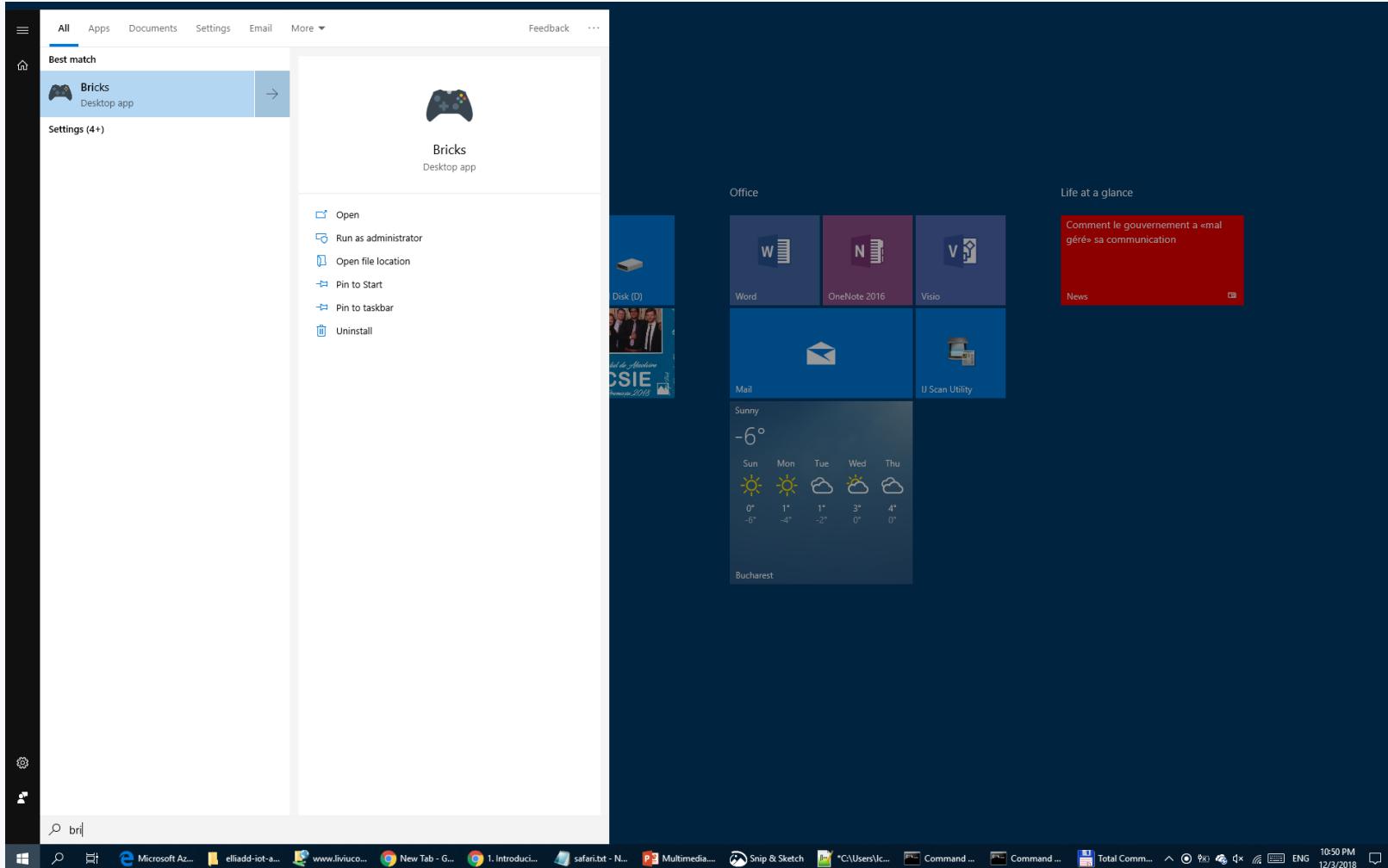
- Using service workers, progressive web apps can launch in an instant, whether the user has a blazing fast connection, an unreliable 2G connection, or even no connection at all. Sites can load in milliseconds, much faster than anything we have experienced on the web before, and sometimes faster than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

# Progressive Web Apps

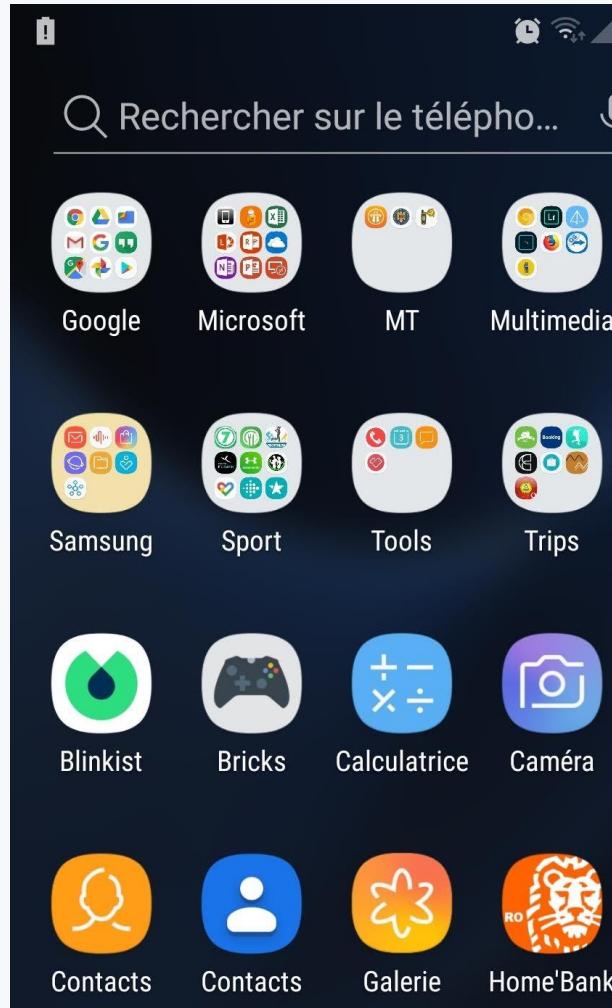
## Engaging



Desktop - Homescreen shortcut

# Progressive Web Apps

## Engaging



Mobile - Homescreen shortcut

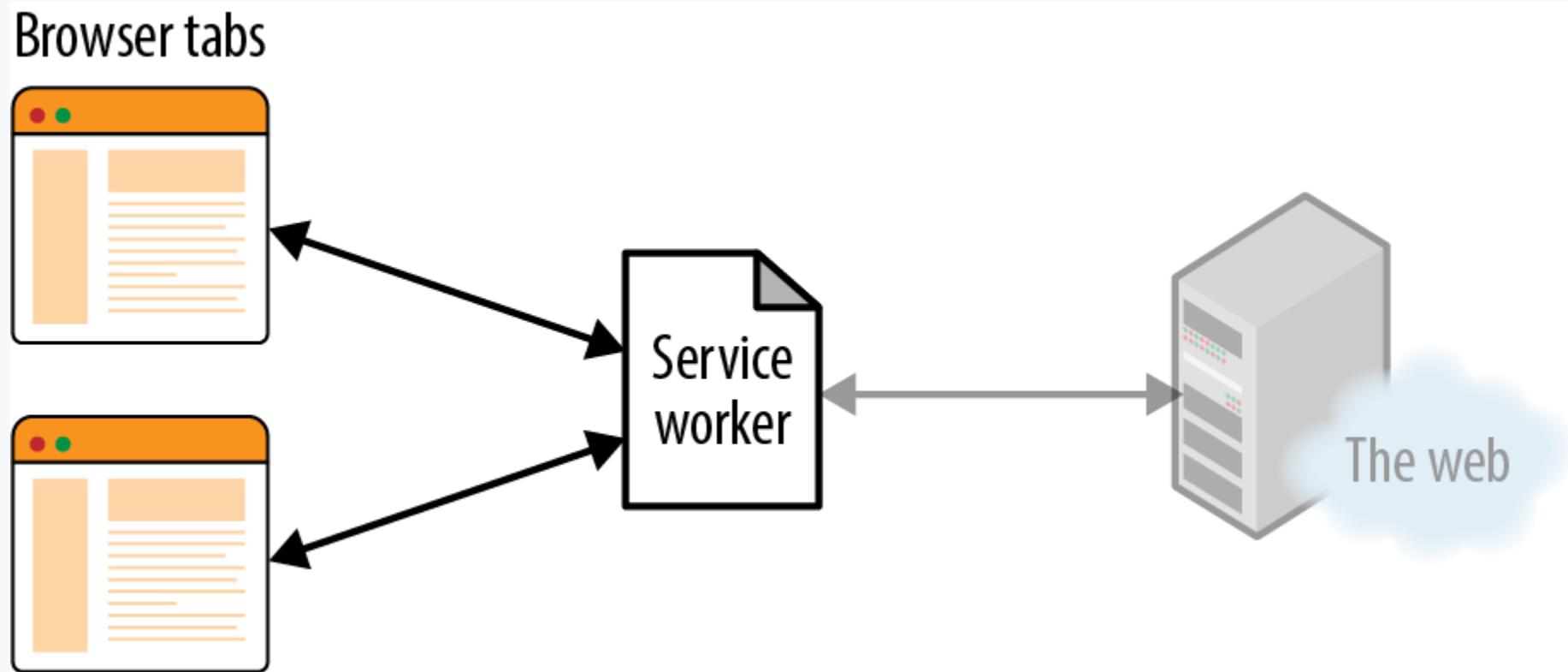
# Engaging

- 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



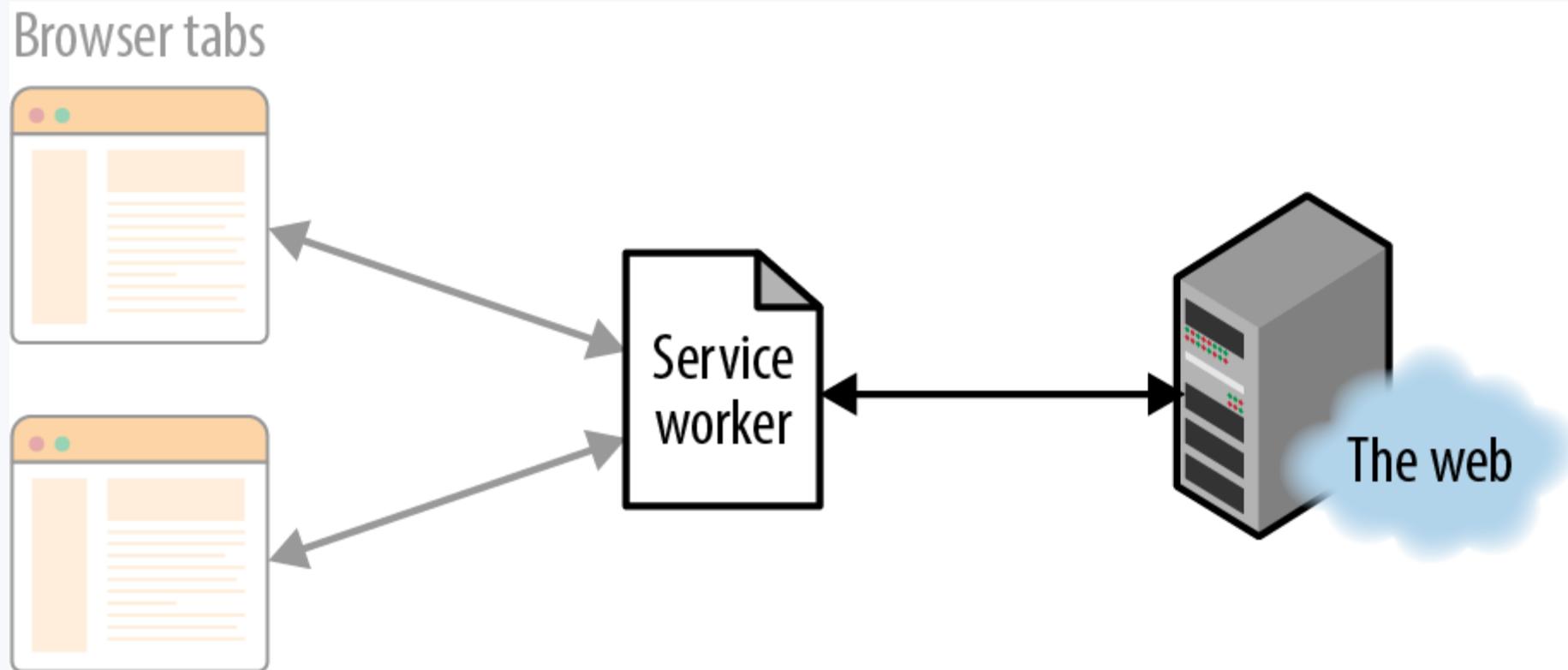
Pages communicating with a service worker while the user is **offline**

# 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

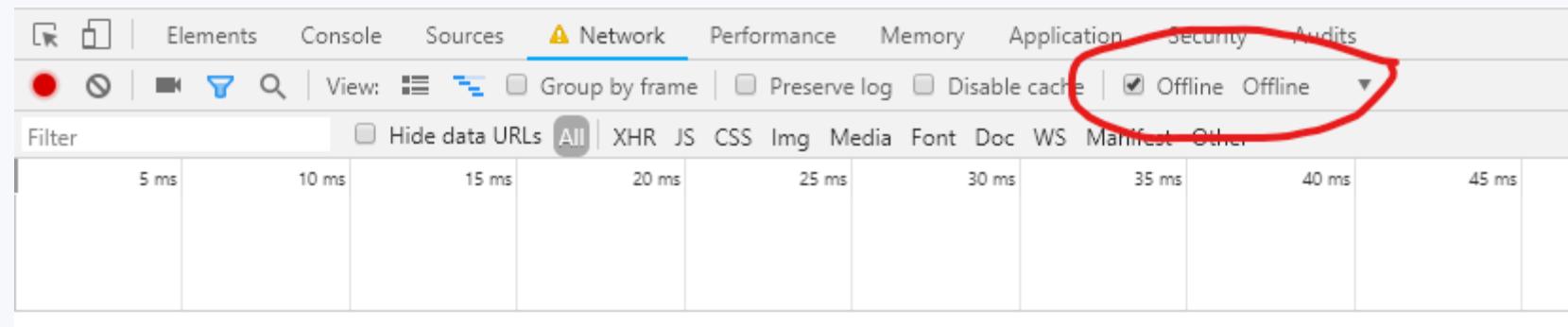
The screenshot shows the 'Service Workers' panel in the Google Chrome DevTools. At the top, there are three checkboxes: 'Offline', 'Update on reload', and 'Bypass for network'. Below this, the address bar shows '127.0.0.1'. Underneath, it says 'Source [serviceworker.js](#) ✖ 8'. To the right of this, the text 'Received 12/11/2018, 11:40:44 AM' is displayed. Below this, the 'Status' section shows a green dot next to '#5335 activated and is running' and a link to 'stop'. Below that, an orange dot is shown next to '#5501 waiting to activate' and a link to 'skipWaiting'. To the right of this, the text 'Received 12/17/2018, 10:49:51 PM' is displayed. Finally, the 'Clients' section lists three entries, each consisting of a URL and a 'focus' link. The URLs are 'http://127.0.0.1:5500/index.html', 'http://127.0.0.1:5500/index.html', and 'http://127.0.0.1:5500/index.html'. The last two URLs are heavily highlighted with yellow.

# 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](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

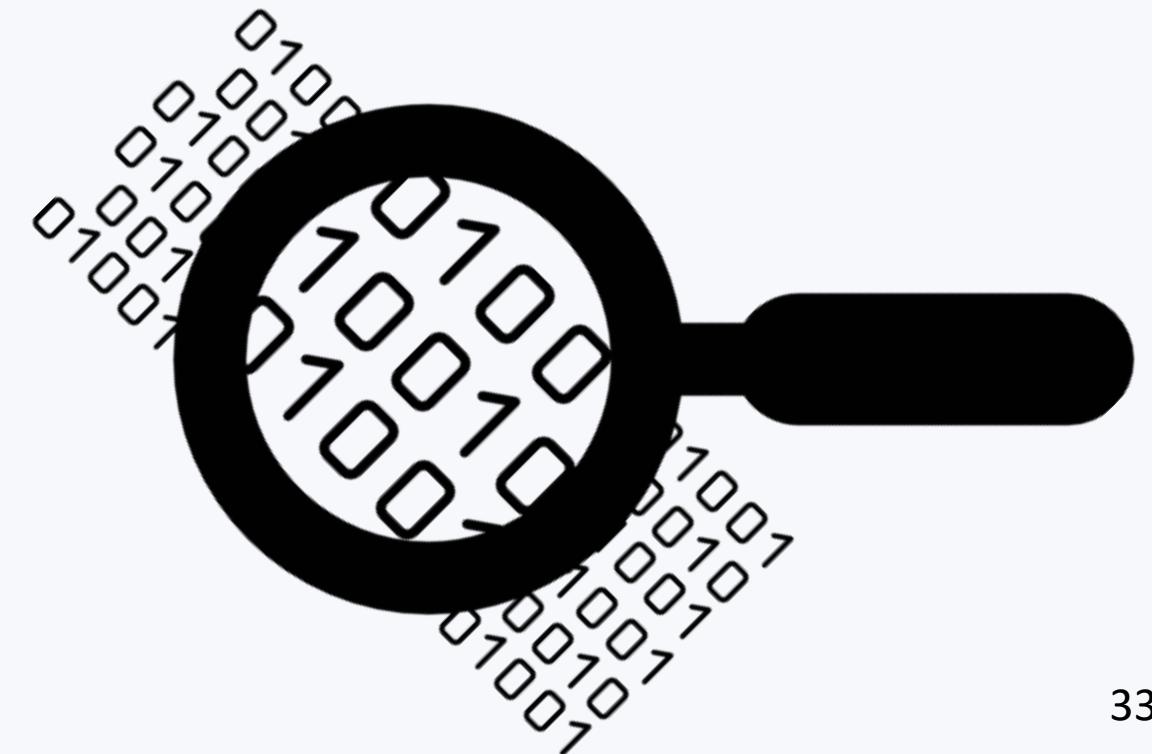
- Intercept requests from the Service Worker

```
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](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.

# Progressive Web Apps

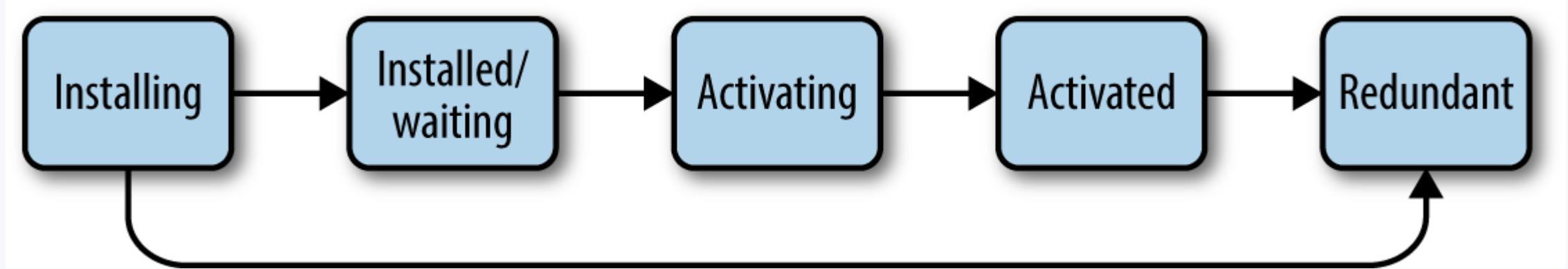
## Service Worker

The screenshot shows the Chrome DevTools interface with the Application tab selected. In the left sidebar under 'Application', 'Service Workers' is selected. The main panel displays information about a service worker registered at '127.0.0.1'. The service worker's source is listed as 'serviceworker.js', received on '12/4/2018, 9:49:11 AM'. Its status is shown as 'activated and is running' with a green dot. A client is listed as 'http://127.0.0.1:5500/index.html'. There are two interaction boxes: one for 'Push' containing 'Test push message from DevTools.' with a 'Push' button, and one for 'Sync' containing 'test-tag-from-devtools' with a 'Sync' button.

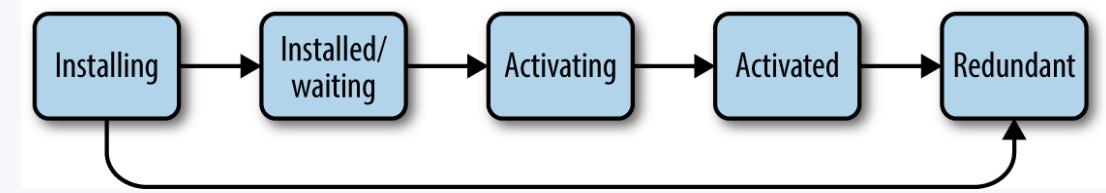
Turning on **Update on reload**

# Service Worker LifeCycle

- The lifecycle of a service worker after being downloaded:

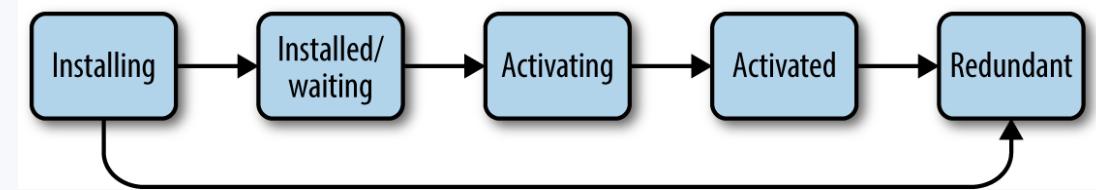


# Service Worker LifeCycle



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

# Service Worker LifeCycle

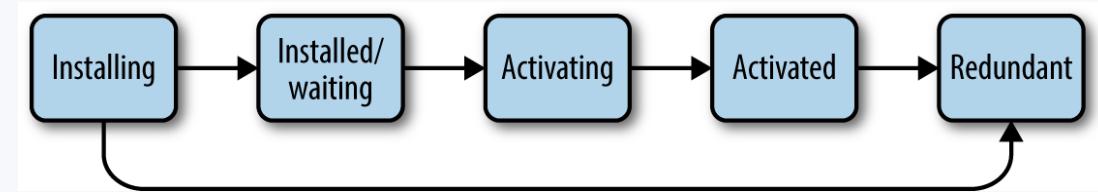


- **Installing state**

- 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

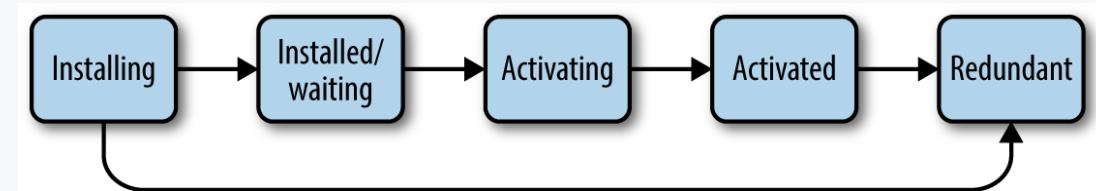


- If an error occurs during installation, the script will instead be moved to the redundant state.

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

A screenshot of the Chrome DevTools Application tab. The tab bar includes Sources, Network, Performance, Memory, Application (which is underlined), Security, and Audits. Under the Application tab, there's a 'Service Workers' section with checkboxes for Offline, Update on reload, and Bypass for network. Below that, a list shows a service worker entry for '127.0.0.1 - deleted'. The entry details are: Source: [serviceworker.js](#) (with a red 'x' icon and the number 10), Received: 12/17/2018, 10:54:46 PM, Status: #5507 is redundant. There are buttons for Push and Sync at the bottom.

# Service Worker LifeCycle



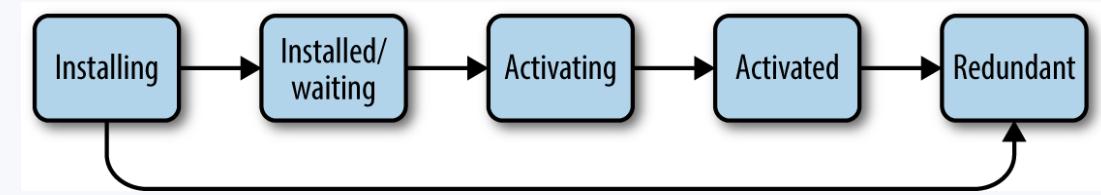
- Installed/waiting state

- 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**.

The screenshot shows the Chrome DevTools Application tab for the URL 127.0.0.1. The left sidebar lists 'Manifest', 'Service Workers' (which is selected), and 'Clear storage'. The main panel shows the 'Service Workers' section with the following details:

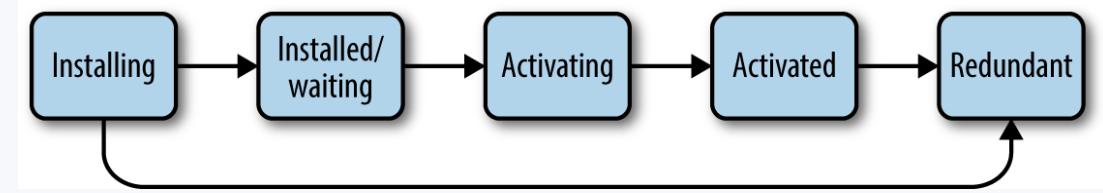
- Source:** [serviceworker.js](#)
- Received:** 12/17/2018, 11:07:59 PM
- Status:** #5509 activated and is running [stop](#)
- #5511** waiting to activate [skipWaiting](#)
- Received:** 12/17/2018, 11:08:15 PM
- Clients:** <http://127.0.0.1:5500/index.html> [focus](#)

# Service Worker LifeCycle



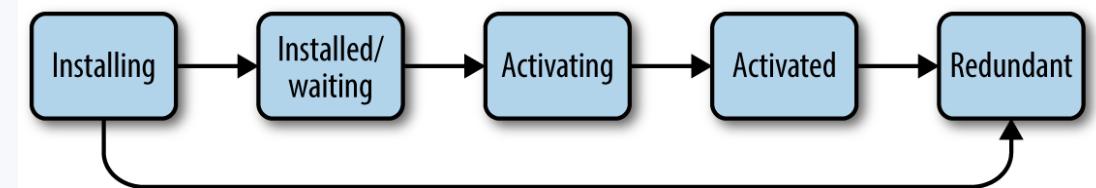
- It is only move to the **activating state** when there are no longer any pages loaded that are still using the old service worker.
- **Why?** Imagine a scenario in which you release a new version of a service worker, and this service worker's install event deletes userdata.json from the cache, adds 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

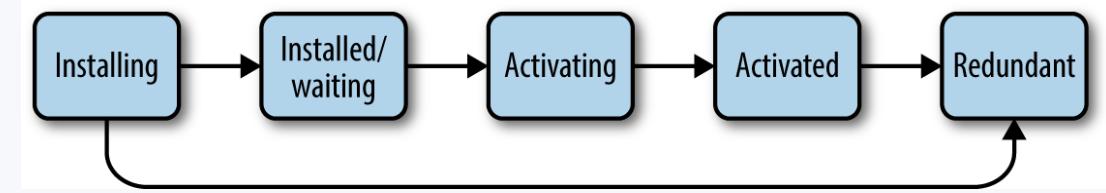


- Activating state

- 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.

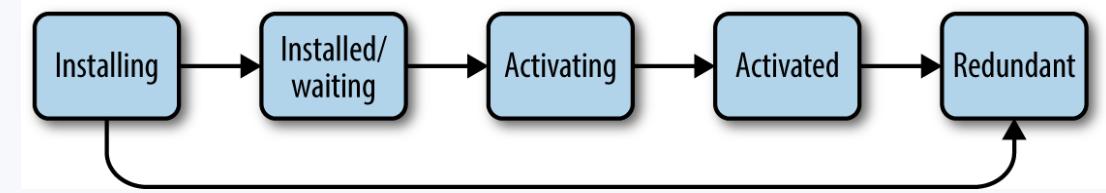
```
self.addEventListener('activate', function(e) {  
  e.waitUntil(  
    caches.keys().then(function(keyList) {  
      return Promise.all(keyList.map(function(key) {  
        if (key !== cacheName) {  
          return caches.delete(key);  
        }  
      }));  
    })  
  );  
  return self.clients.claim();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

# 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.`);
});
```

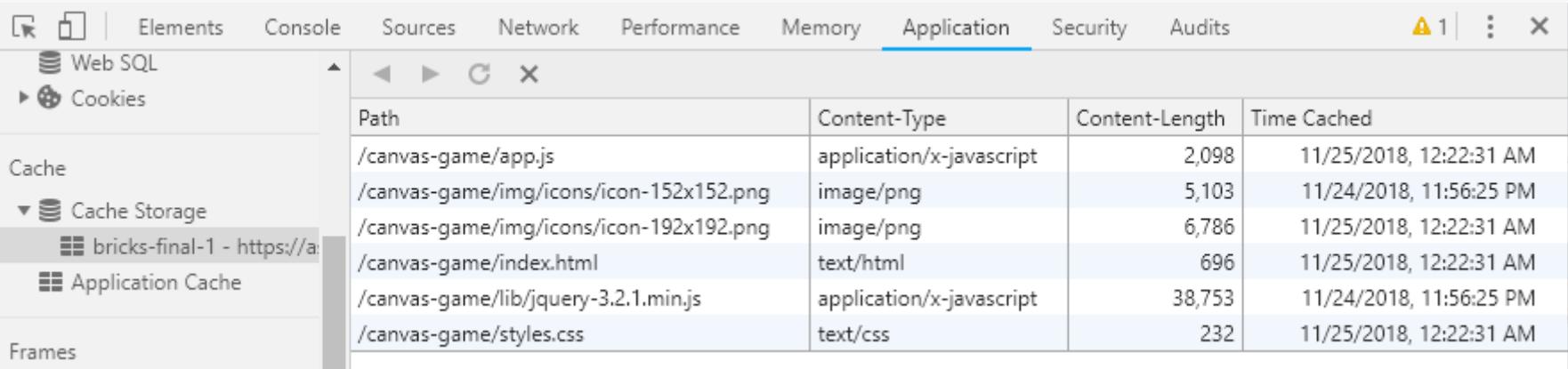
# CacheStorage API

- 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'
];
```



The screenshot shows the Google Chrome DevTools Application tab. On the left, there's a sidebar with icons for Web SQL, Cookies, Cache, Cache Storage (which is expanded to show 'bricks-final-1 - https://a...', and Application Cache), and Frames. The main area has tabs for Path, Content-Type, Content-Length, and Time Cached. The table lists the cached files from the provided code:

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)
```

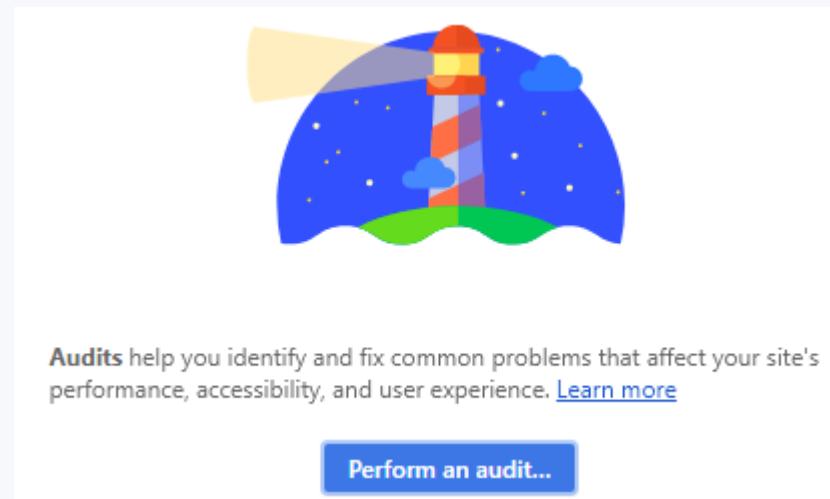
- The function returns a Promise that resolves to true if the cache existed and was deleted, or false otherwise.

# 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();
});
```

# 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

- 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>

# Examples

Drawing- Google Chrome Canvas

- <https://canvas.apps.chrome/>

# JavaScript

# JavaScript

- Modern\_JavaScript.zip on <http://online.ase.ro>
- <http://jstherightway.org/>

# Web Storage API

- Documentation:
  - [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Storage\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API)