Tokaj-Hegyalja Egyetem

Komputer grafika és képszerkesztés

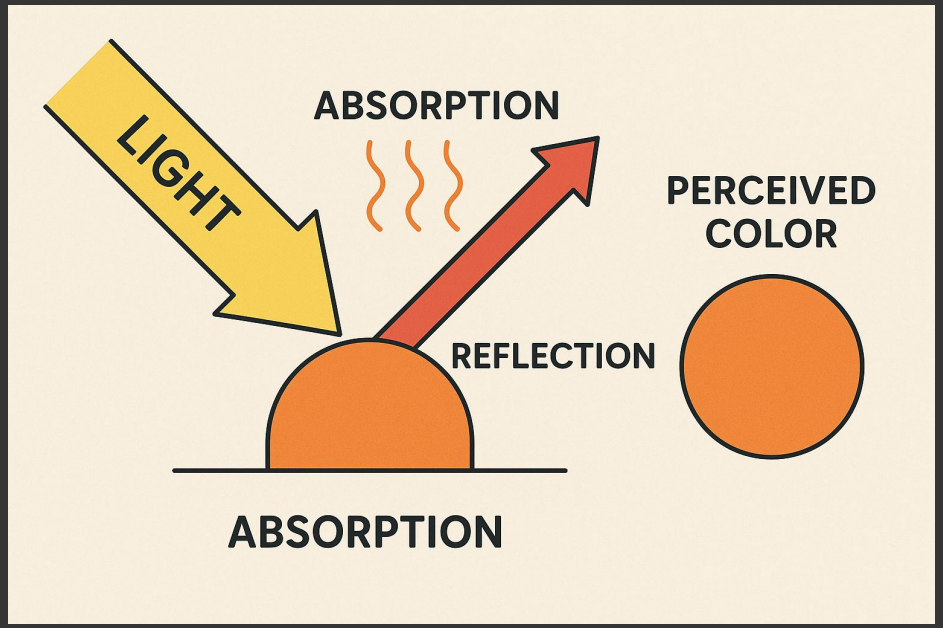
3. előadás

Mileff Péter - Miskolci Egyetem

Színmodellek

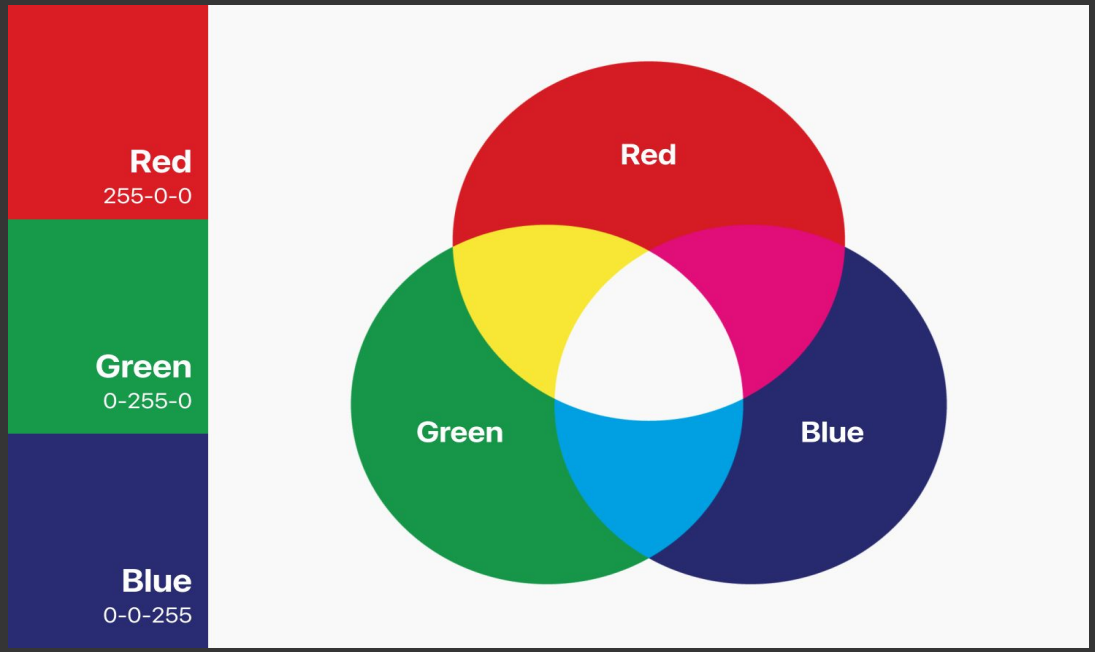
- **A színmodellek célja, hogy a színeket egyértelműen, numerikusan leírható formában reprezentálják.**
- A számítógépes rendszerek nem „színeket” kezelnek, hanem számokat
  - Ezért szükség van olyan matematikai modellekre, amelyek a vizuális színélményt számértékek halmazaként írják le.
- Egy színmodell tehát nem más, mint egy koordinátarendszer, amelyben minden pont egy adott színnek feleltethető meg.
- A legelterjedtebb színmodell a digitális megjelenítésben az **RGB** modell.
- Az RGB színmodell az additív színkeverés elvén alapul, ahol a **vörös**, **zöld** és **kék** alapszínek különböző intenzitású kombinációi hozzák létre a többi színt.
- Ez a modell közvetlenül illeszkedik a kijelzők működéséhez
  - a monitorok és egyéb digitális megjelenítők is vörös, zöld és kék fényt kibocsátó alpixelek segítségével állítják elő a képet.

Színmodellek



RGB modell

- Sajátossága, hogy az alapszínek maximális intenzitású összeadása fehér színt eredményez,
  - míg az összes komponens nullára állítása fekete képet ad



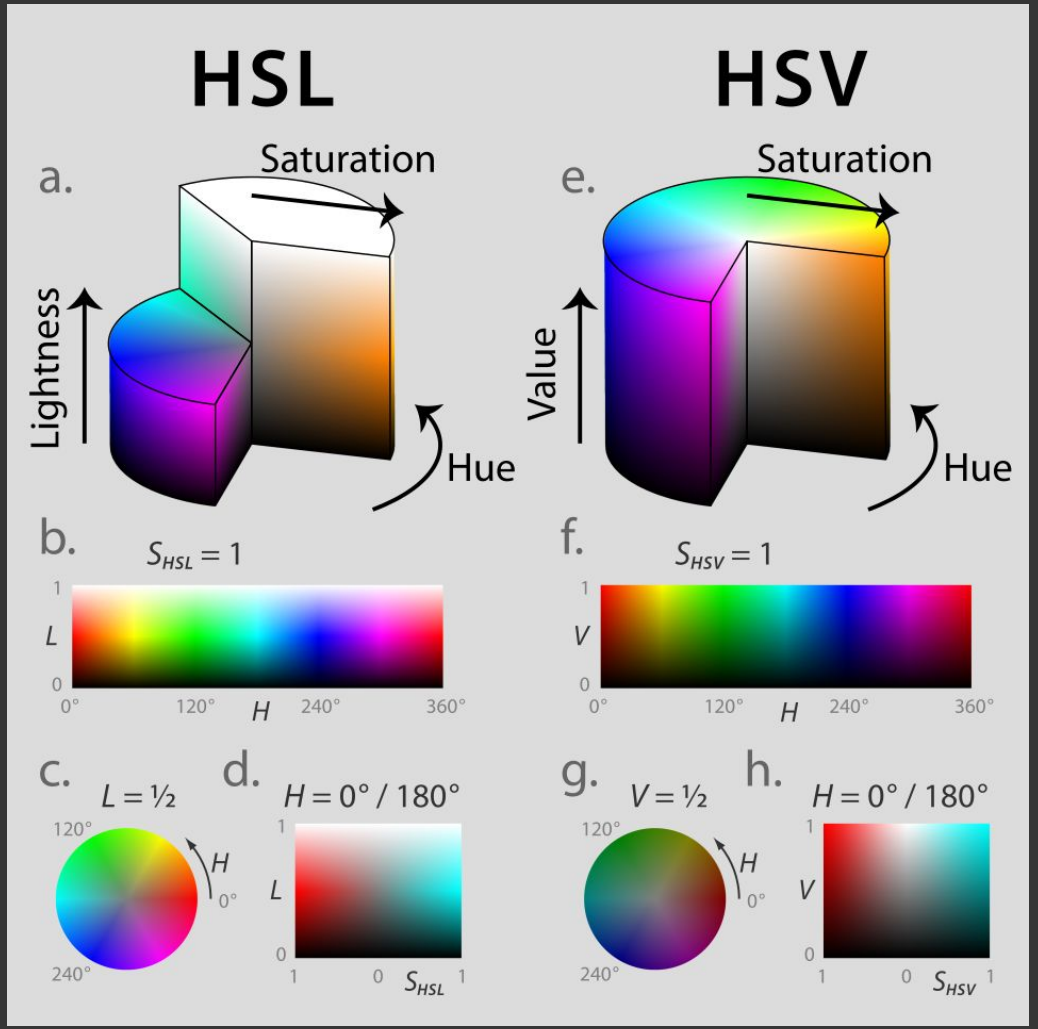
Színmodellek

- **Az RGB modell intuitív a hardveres megjelenítés szempontjából, ugyanakkor kevésbé alkalmas bizonyos képfeldolgozási feladatokra.**
- **Oka:**
  - a három komponens erősen összefügg egymással,
  - a fényerő, illetve a színárnyalat nem választható szét egyértelműen
- Emiatt alakultak ki olyan színmodellek, amelyek a színélményt az emberi érzékeléshez közelebb álló paraméterekre bontják.
- Ezen modellek közé tartozik a **HSV** és a **HSL** színmodell, amelyek a színt három összetevővel írják le:
  - árnyalat
  - telítettség
  - fényesség vagy világosság

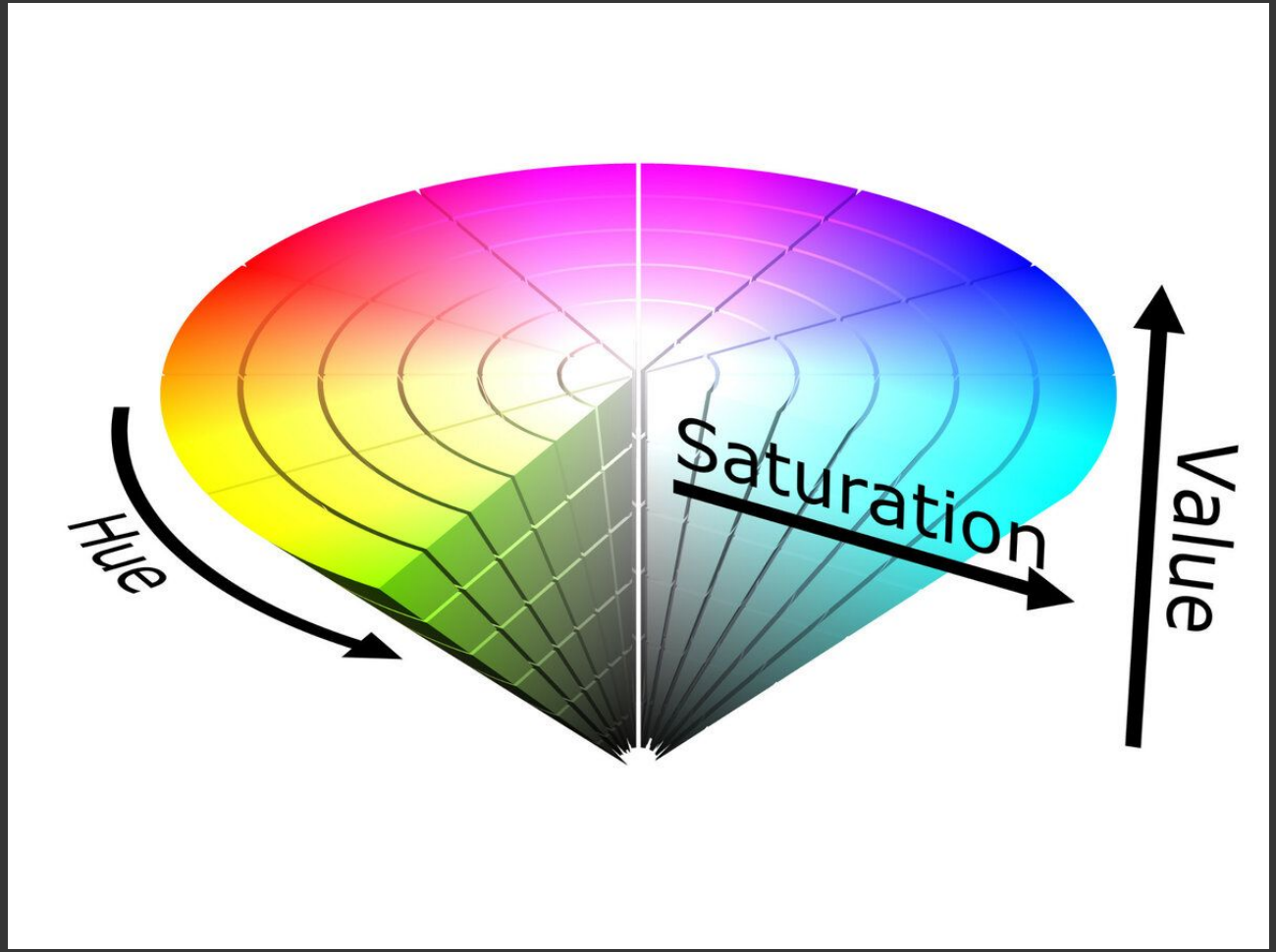
Színmodellek

- **Az árnyalat a szín „típusát” jelöli:**
  - pl vörös, zöld vagy kék,
  - jellemzően körkörös skálán értelmezhető.
- A **telítettség** azt fejezi ki, hogy a szín mennyire élénk vagy mennyire szürkébe hajló.
- A **fényesség** vagy világosság pedig azt határozza meg, hogy a szín mennyire világos vagy sötét.
- Hol hasznos ez?
  - olyan alkalmazásokban, ahol a színek módosítása emberi intuíció szerint történik
    - például grafikai szerkesztőprogramokban vagy felhasználói interfészekben.

HSL, HSV modellek



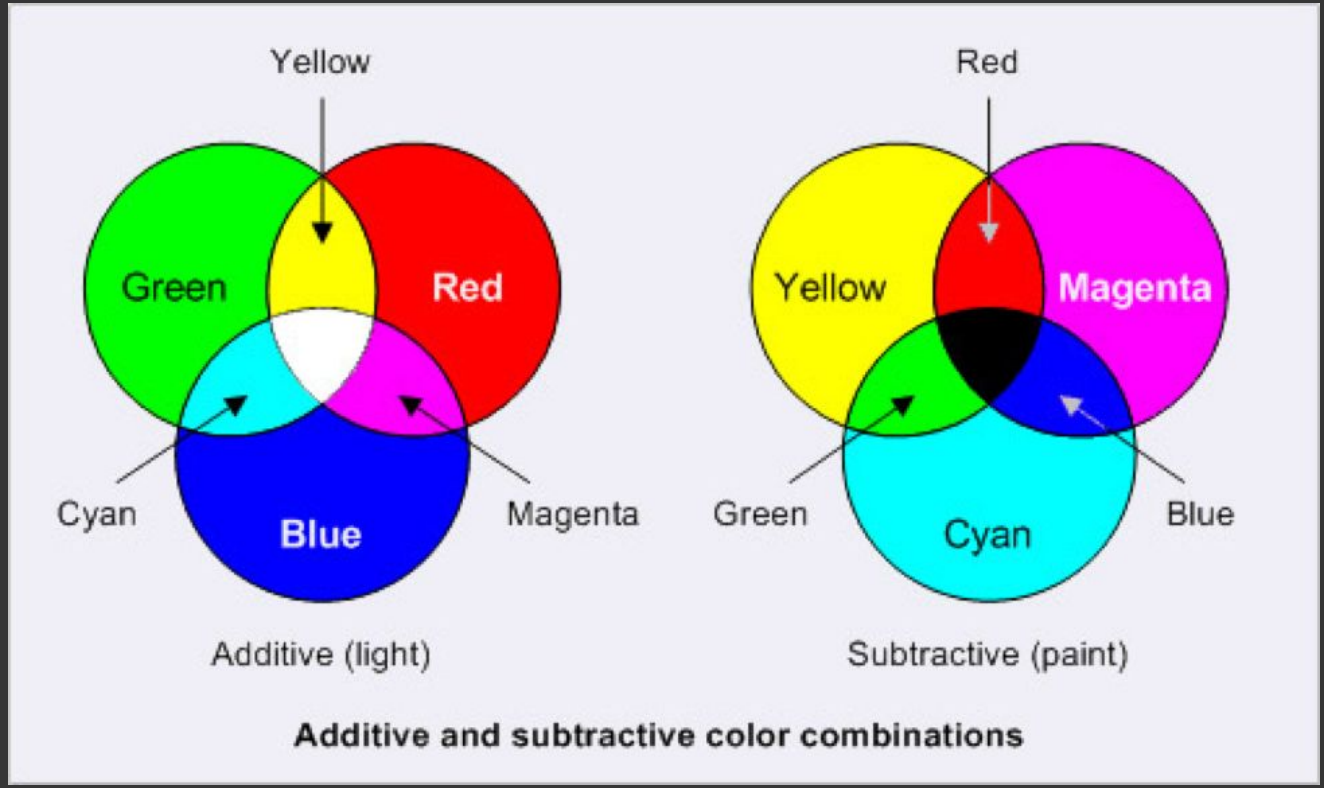
HSL, HSV modellek



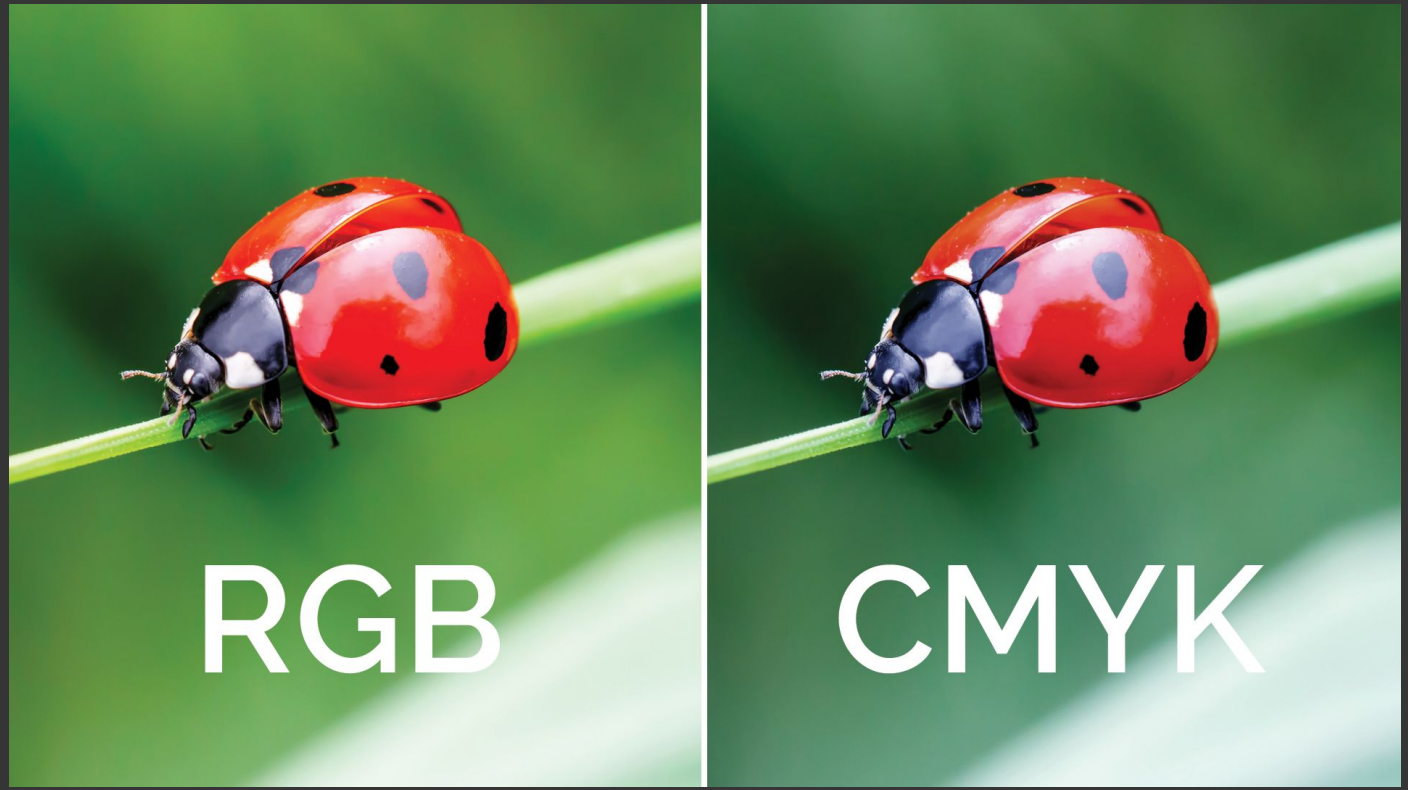
CYMK modell

- A nyomdai és nyomtatási technológiák eltérő fizikai elven működnek, ezért ott más színmodell terjedt el.
- A **CMY** és **CMYK** színmodellek a szubtraktív színkeverés elvén alapulnak
  - ahol a színek létrehozása a fény elnyelésével történik.
- A cián, bíbor és sárga festékek elnyelik a fehér fény egyes komponenseit, így hozzák létre a kívánt színeket.
- A fekete komponens hozzáadása a gazdaságosabb festékhasználat és a mélyebb fekete árnyalat elérése miatt vált szükségessé.
- Ez a színmodell elsősorban nyomtatási folyamatokban használatos, és közvetlenül nem alkalmas kijelzők meghajtására.

CMMK vs RGB


Yellow
Green
Red
Cyan
Blue
Magenta
Additive (light)
Red
Yellow
Magenta
Green
Cyan
Blue
Subtractive (paint)
Additive and subtractive color combinations

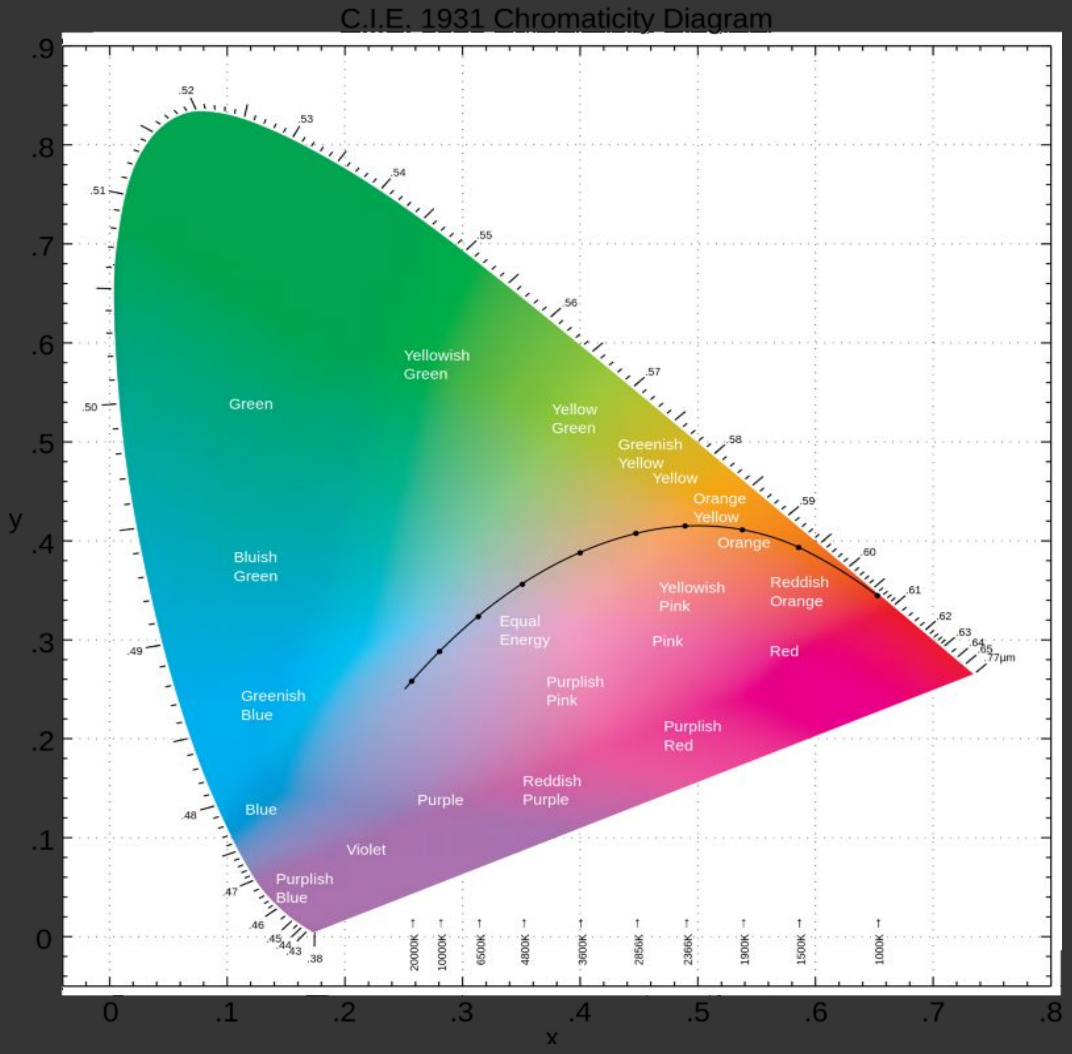
CMMK vs RGB



Általános modellek

- A színmodellek közül külön kategóriát képviselnek az úgynevezett perceptuális vagy eszközfüggetlen modellek.
- Ilyen például a **CIE XYZ** és a **CIE Lab** színmodell
- **Céljuk:** a színek közötti különbségek jobban megfeleljenek az emberi látás érzékenységének.
- Ezekben a modellekben az azonos távolságú pontok nagyjából azonos mértékű színkülönbséget jelentenek a szem számára.
- Ez különösen fontos a színkorrekció, a színkalibráció és a képfeldolgozás területén.

CIE modell


C.I.E. 1931 Chromaticity Diagram
Yellowish Green
Green
Yellow Green
Greenish Yellow
Yellow
Orange Yellow
Orange
Bluish Green
Yellowish Pink
Reddish Orange
Equal Energy
Pink
Red
Purplish Pink
Greenish Blue
Purplish Red
Reddish Purple
Blue
Purple
Violet
Purplish Blue
y
x

Hardveres vs szoftveres rendszerek...

Hardveres és szoftveres rendszerek

- A képek előállítása és megjelenítése hardveres és szoftveres komponensek együttműködésén alapul.
- A grafikus hardver központi eleme a grafikus feldolgozóegység, vagyis a GPU.
- A GPU olyan speciális processzor, amelyet kifejezetten nagyszámú, egyszerű számítás párhuzamos végrehajtására terveztek.
- A központi feldolgozóegység, a **CPU**, viszonylag kevés, de összetett feladatot képes hatékonyan végrehajtani,
- A **GPU** több ezer kisebb feldolgozóegységgel rendelkezik, amelyek egyszerre dolgoznak az adatokon.
- Ez a felépítés különösen alkalmassá teszi a grafikai számítások elvégzésére, ahol ugyanazt a műveletet kell sok adaton végrehajtani,
    - például pixelek vagy vertex-ek esetén

Hardveres és szoftveres rendszerek

- A grafikus hardver szorosan együttműködik a memóriarendszerrel.
- A GPU saját memóriával rendelkezik, amelyben a textúrák, képkockák és egyéb grafikai adatok kerülnek tárolásra.
- A memória sávszélessége és késleltetése jelentős hatással van a grafikus alkalmazások teljesítményére,
    - mivel a feldolgozóegységeknek folyamatosan nagy mennyiségű adatot kell elérniük.
- A modern grafikus rendszerek ezért különösen nagy hangsúlyt fektetnek a memóriahatékonyságra és az adatok párhuzamos elérésére.

Hardveres és szoftveres rendszerek

- A szoftveres grafikus rendszerek szerepe abban áll, hogy kapcsolatot teremtsenek az alkalmazás és a grafikus hardver között.
- Egy alkalmazás közvetlenül nem fér hozzá a GPU alacsony szintű működéséhez,
  - ezért grafikus programozási felületeket, úgynevezett **grafikus API-kat használ**
- Ezek az API-k egységes módot biztosítanak a grafikai műveletek leírására, miközben elrejtik a hardver specifikus részleteket.
- Ennek köszönhetően ugyanaz az alkalmazás különböző grafikus hardvereken is futtatható.

Hardveres és szoftveres rendszerek

- A grafikus API-k feladata nem csupán a parancsok továbbítása a GPU felé
  - hanem a grafikus pipeline logikai felépítésének meghatározása is
- Az alkalmazás ezek segítségével adja meg:
  - milyen objektumokat szeretne megjeleníteni,
  - milyen transzformációkat kell alkalmazni,
  - hogyan történjen az árnyalás.
- A szoftveres réteg felel továbbá az erőforrások kezeléséért
  - például a memóriában tárolt textúrák és pufferek létrehozásáért és felszabadításáért.

Hardveres és szoftveres rendszerek

- A hardveres gyorsítással szemben léteznek tisztán szoftveres grafikus megoldások is,
  - ahol a teljes rajzolási folyamat a CPU-n fut.
  - **Software rendering / Software rasterisation**
- Ezek a rendszerek jellemzően lassabbak, azonban oktatási és kísérleti célokra rendkívül hasznosak.
- A szoftveres grafika lehetővé teszi, hogy a grafikai algoritmusok működése lépésről lépésre megérthető legyen,
    - mivel minden számítás expliciten a programkódban történik.
- Különösen fontos a pixel szintű rajzolás és az alapvető raszterizációs algoritmusok elsajátításakor.

Képernyő - pixel - koordináta rendszer...

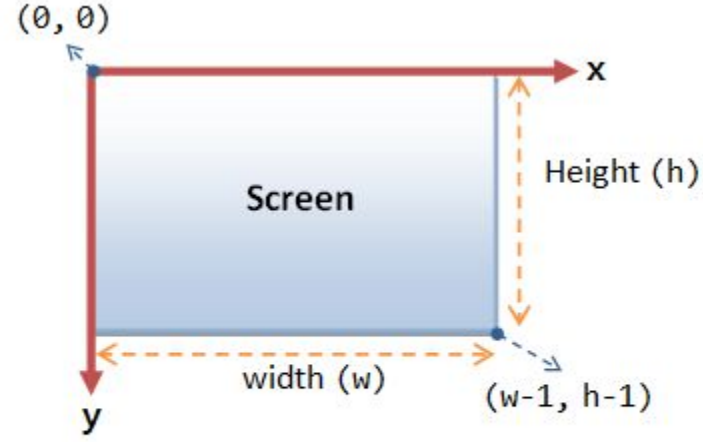
Képernyő - Pixelek

- A modern kijelzők raszteres elven működnek, vagyis a megjelenített kép képpontok szabályos rácsából áll.
- Ezek a képpontok, vagy **pixel**ek, a megjelenítés legkisebb önálló egységei, amelyek egy adott helyen meghatározott színnel világítanak.
- A képernyő felbontása azt adja meg, hogy vízszintes és függőleges irányban hány ilyen pixel található.
- Függetlenül attól, hogy egy jelenet bonyolult háromdimenziós modellből vagy egyszerű geometriai alakzatokból áll,
    - a grafikus motornak minden képkocka esetén meg kell határoznia az összes érintett pixel színét.

Képernyő - Pixelek

- A pixelek elhelyezkedésének leírásához koordinátarendszert használunk.
- A képernyő koordinátarendszere jellemzően kétdimenziós, és minden pixel egy egyedi koordinátapárral azonosítható.
- A koordináták értelmezése azonban nem minden grafikus rendszerben azonos.
- **Gyakori megoldás:**
    - a koordinátarendszer origója a képernyő bal felső sarkában található
    - a vízszintes tengely jobbra, a függőleges tengely pedig lefelé növekszik
    - Ez a választás technikai szempontból praktikus, mivel illeszkedik a képmemória sorfolytonos tárolásához.

Koordináta rendszer



The 2D Screen Coordinates: The origin is located at the top-left corner, with x-axis pointing left and y-axis pointing down.

Képernyő - Pixelek

- Más rendszerekben az origó a bal alsó sarokban helyezkedik el
  - a függőleges tengely felfelé mutat.
- Ez a megközelítés közelebb áll a matematikában megszokott koordinátarendszerhez,
  - gyakran alkalmazzák olyan grafikus környezetekben, ahol a geometriai szemlélet hangsúlyosabb.
- Mivel a pixelek diszkrét helyeken találhatók, egy képpont pozícióját tipikusan egész koordináták írják le,
  - a grafikai algoritmusok ezért dolgoznak gyakran egész számokkal

Pixel szintű rajzolás

- **A pixel szintű rajzolás a komputergrafika legalsó, legkonkrétabb szintje**
  - a grafikai műveletek közvetlenül a képernyőn megjelenő képpontokra hatnak
  - különösen fontos az alapfogalmak megértéséhez
- Egy vonal, egy kör vagy bármilyen alakzat matematikai értelemben folytonos objektum
  - azonban a képernyőn csak közelítő módon jeleníthető meg pixelek segítségével.
  - A rajzolás során tehát minden esetben döntést kell hozni arról, hogy egy adott pixel a rajzolt alakzat részének tekinthető-e vagy sem.
  - Ez a döntési folyamat adja a rasztergrafika lényegét.
- A grafikus rajzolás atomi egysége: pixel beállítása egy adott színre

Pixel szintű rajzolás

- Minden bonyolultabb rajzolási algoritmus ilyen egyszerű pixelmódosítások sorozatára bontható
    - legyen szó vonalról, sokszögről vagy képfeldolgozási műveletről, végső soron.
- Külön figyelmet kell fordítani a koordináták kezelésére:
    - Mivel a pixelek rácsszerű elrendezésben helyezkednek el, a rajzolás során jellemzően egész számú koordinátákkal dolgozunk.
- Ez azonban felveti a kérdést: miként ábrázolhatók olyan alakzatok, amelyek matematikai leírása nem illeszkedik pontosan az egész koordinátákhoz.
    - A grafikai algoritmusok feladata ilyenkor az, hogy a lehető legjobb közelítést adják a rendelkezésre álló pixelek segítségével

Pixel szintű rajzolás

- Ez a közelítés szükségszerűen vizuális kompromisszumokkal jár
- A ferde vagy ívelt vonalak lépcsőzetes hatást mutathatnak,
  - a pixelek négyzetes rácsa nem képes pontosan követni a folytonos görbéket.
  - Ezt a jelenséget lépcsőzetességnek vagy **aliasing**nek nevezzük.
- Fontos: a vizuális hibák nem a megvalósítás „rosszaságából” fakadnak, hanem a diszkrét megjelenítés természetes következményei.

Pixel szintű rajzolás

- **Egy rajzoló algoritmusnak hatékonynak kell lennie**
  - valós idejű grafika esetében akár másodpercenként több millió pixel állapotát kell meghatározni
  - Ezért az egyszerű, gyors számításokra épülő algoritmusok kiemelt szerepet kapnak,
    - különösen olyan környezetekben, ahol a hardveres gyorsítás nem érhető el vagy nem használható.
- Képmemória szerepe:
  - A képernyőn megjelenő kép jellemzően egy memóriaterületen, úgynevezett **framebuffer**ben kerül tárolásra
  - minden pixelhez tartozik egy vagy több érték, amelyek a színét írják le.
  - A rajzolás ennek a memóriaterületnek a módosítását jelenti.

Vonal rajzolás...

Vonal rajzolás - DDA

- A vonalrajzolás problémája a raszteres megjelenítés egyik klasszikus és alapvető kérdése
    - mivel a vonal matematikailag folytonos objektum, míg a képernyő pixelekből álló diszkrét rács.
- A **DDA**, vagyis **Digital Differential Analyzer** algoritmus az egyik legegyszerűbb és legérthetőbb módszer.
- Az algoritmus célja:
    - két pont által meghatározott egyenes szakaszt úgy jelenítsen meg a képernyőn, hogy a kiválasztott pixelek vizuálisan a lehető legjobban kövessék a matematikai vonalat.

Vonal rajzolás - DDA

- Egyenes egyenlete: $y = mx + c$, ahol

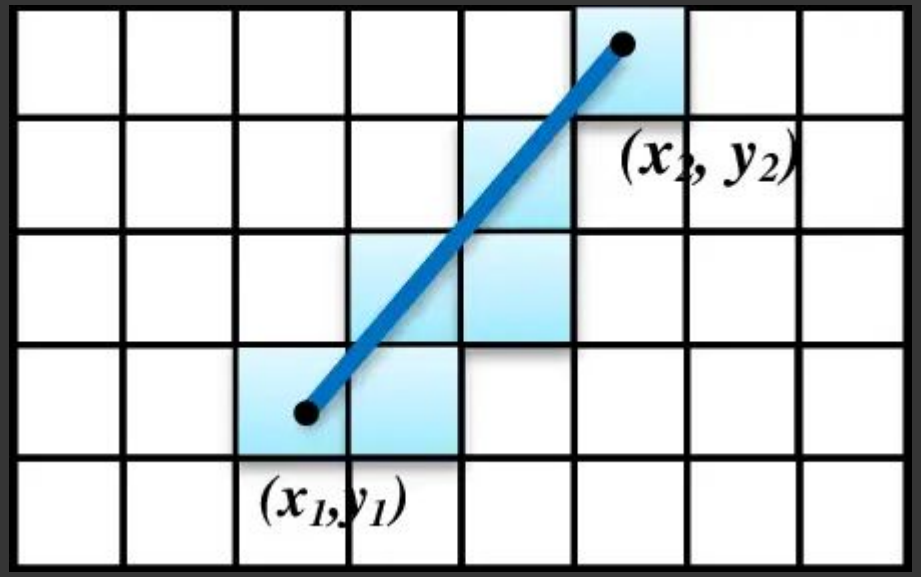
m a vonal meredeksége,
c pedig a y tengely értéke, amikor x = 0

- A DDA algoritmus alapötlete az **inkrementális számítás**:
  - Ahelyett, hogy minden pixelhez újra kiszámítanánk az egyenes egyenletét, az algoritmus a vonal egyik végpontjából indul ki,
    - majd lépésről lépésre halad a másik végpont felé.
  - Minden lépésben az aktuális pozícióhoz hozzáad egy előre meghatározott, kis mértékű eltolást, amely a vonal irányát követi.
  - Ez a megközelítés jól illeszkedik a pixel szintű rajzolás szemléletéhez, mivel a folyamat során egy pixelt rajzolunk ki, majd kiszámítjuk a következő pixel helyét.

Vonal rajzolás - DDA

- A lépések irányának és nagyságának meghatározásához:
  - az algoritmus megvizsgálja a két végpont közötti különbséget vízszintes és függőleges irányban.
  - A nagyobb abszolút értékű eltérés határozza meg, hogy hány lépésben rajzoljuk ki a vonalat.
  - Ez biztosítja, hogy a vonal minden esetben összefüggőnek tűnjön, függetlenül attól, hogy meredek vagy lapos egyenesről van szó.
  - A másik irányban a lépés nagysága ennek megfelelően arányosan kerül meghatározásra.

Vonal rajzolás - DDA



Vonal rajzolás - DDA

Lépései:

1. Különbségek számítása: Határozza meg a két végpont közötti különbséget: dx=x2-x1, dy=y2-y1.
2. Lépésszám meghatározása: Határozza meg a lépések számát a nagyobbik különbség alapján: ha |dx| > |dy|, akkor lépések = |dx|, különben steps = |dy|.
3. Növekmények kiszámítása: Számítsa ki az x és y irányú növekményt:

   x_increment = dx / steps,
   y_increment = dy / steps.
4. Pixel rajzolás: Kezdőponttól (x1,y1) kezdve, adjon hozzá növekményeket:

   x=x + x_increment
   y=y + y_increment.

   Kerekítse a koordinátákat a legközelebbi egész számra a pixel beállításához.
5. Ciklus: Ismételje a 4. lépést, amíg el nem éri a végpontot.

Vonal rajzolás - DDA

DDA Algoritmus C kód:

```
void DDA(int X0, int Y0, int X1, int Y1) {
    // calculate dx & dy
    int dx = X1 - X0;
    int dy = Y1 - Y0;

    // calculate steps required for generating pixels
    int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);

    // calculate increment in x & y for each steps
    float Xinc = dx / (float)steps;
    float Yinc = dy / (float)steps;

    // Put pixel for each step
    float X = X0;
    float Y = Y0;

    for (int i = 0; i <= steps; i++) {
        putpixel(round(X), round(Y),RED); // put pixel at (X,Y)
        X += Xinc; // increment in x at each step
        Y += Yinc; // increment in y at each step
    }
}
```

Vonal rajzolás - DDA

DDA algoritmus hátrányai:

- A lebegőpontos aritmetika használata: A DDA algoritmus lebegőpontos számításokat igényel, amelyek egyes rendszereken lassúak lehetnek.
    - Különösen problémát jelenthet nagy mennyiségű adat feldolgozásakor.
- Korlátozott pontosság: A lebegőpontos aritmetika alkalmazása bizonyos esetekben pontossági korlátokhoz vezethet, különösen akkor, ha az egyenes meredeksége nagyon nagy vagy nagyon kicsi.
- Kerekítési hibák: A számítások során kerekítési hibák léphetnek fel, amelyek pontatlanságot okozhatnak a kirajzolt vonalban. Ez különösen akkor jellemző, amikor az egyenes meredeksége közel van az 1-hez.
- Függőleges egyenesek kezelése: A DDA algoritmus nem képes közvetlenül kezelni a függőleges egyeneseket, mivel ilyen esetben a meredekség nem értelmezhető

Vonal rajzolás - DDA

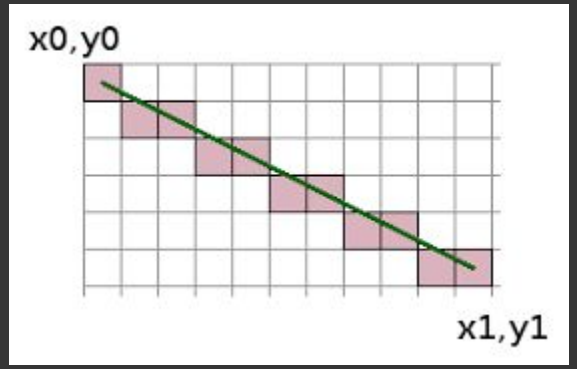
DDA algoritmus hátrányai:

- Nem hatékony összetett görbék esetén: nem alkalmas összetett görbék, például körök és ellipszisek rajzolására, mivel ezek pontos közelítéséhez nagyszámú egyenes szakasz szükséges.
- Aliasing jelenség: Aliasing akkor lép fel, amikor a DDA algoritmussal generált vonalszakaszok nem adják vissza pontosan a rajzolt egyenest, ami a vonal recézett, lépcsőzetes megjelenését eredményezi.
- Vastag vonalak rajzolásának problémái: Az algoritmus alapvetően vékony vonalakat generál, ami problémát okozhat vastag vonalak rajzolásakor
  - mivel az egyes vonalszegmensek átfedhetik egymást, vagy éppen hézagok maradhatnak közöttük.

Bresenham-algoritmus

- A **Bresenham-algoritmus** a raszteres vonalrajzolás egyik legismertebb és legfontosabb módszere
  - kifejezetten a DDA algoritmus hiányosságainak kiküszöbölésére fejlesztettek ki.
- Legfontosabb újítása: a rajzolás során kizárólag egész számokkal dolgozik.
  - Ez jelentős előnyt jelent a lebegőpontos aritmetikát használó DDA-hoz képest,
  - különösen olyan rendszerekben, ahol a lebegőpontos műveletek lassúak vagy korlátozottak.
- **Az algoritmus alapgondolata**: minden lépésben csak egy döntést kell meghozni:
  - a következő pixel közül melyik fekszik közelebb az ideális matematikai egyeneshez.

Bresenham-algoritmus



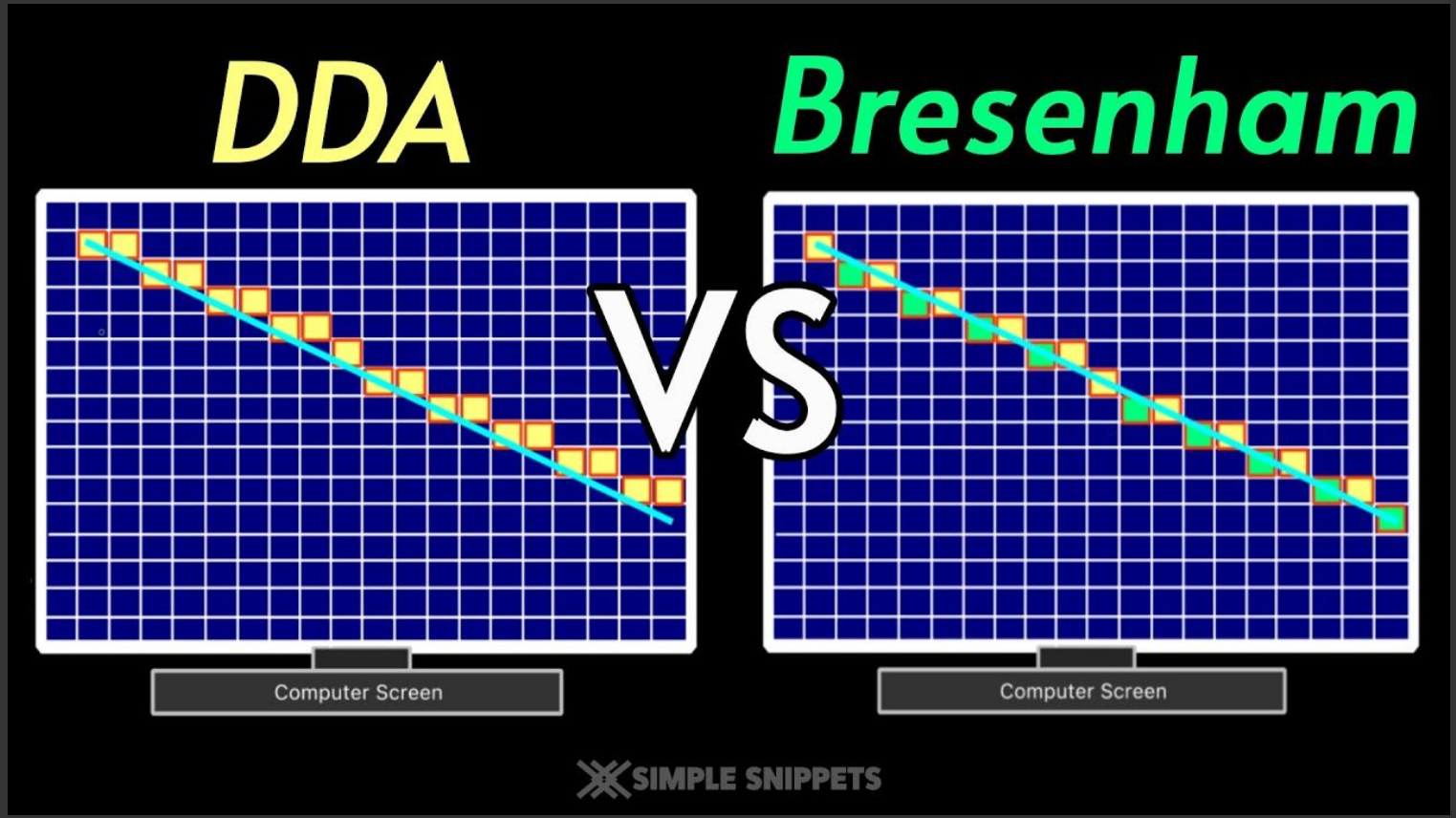
Bresenham-algoritmus

```
// function for line generation
void bresenham(int x1, int y1, int x2, int y2)
{
    int m_new = 2 * (y2 - y1);
    int slope_error_new = m_new - (x2 - x1);
    for (int x = x1, y = y1; x <= x2; x++) {
        cout << "(" << x << "," << y << ")\n";

        // Add slope to increment angle formed
        slope_error_new += m_new;

        // Slope error reached limit, time to
        // increment y and update slope error.
        if (slope_error_new >= 0) {
            y++;
            slope_error_new -= 2 * (x2 - x1);
        }
    }
}
```

Bresenham-algoritmus

- A Bresenham-algoritmus egyik nagy előnye, hogy stabil és kiszámítható eredményt ad
- Mivel nem használ lebegőpontos számításokat:
  - a kerekítési hibák nem halmozódnak fel, és a vonal megjelenése minden esetben konzisztens marad.
  - Ez különösen fontos akkor, amikor sok vonalat kell rajzolni, vagy amikor a grafikus megjelenítésnek valós időben kell működnie.
- Az algoritmus további előnye, hogy képes kezelni a különböző meredekségű egyeneseket beleértve a meredek és lapos vonalakat is.

Bresenham vs DDA



Bresenham-algoritmus

- **A vonalrajzolás során az algoritmus egy domináns irányt választ**
  - amely mentén a rajzolás minden lépésben biztosan halad.
  - Ez jellemzően az az irány, amelyben a két végpont közötti eltérés nagyobb.
- Minden lépésben az algoritmus egy döntési változót frissít:
  - amely azt jelzi, hogy az aktuális pixel után a vonal inkább a szomszédos pixel irányába térjen el felfelé vagy lefelé.
  - A döntés tisztán egész aritmetikán alapul, összeadások és kivonások segítségével.
- **A döntési mechanizmus lényege:**
  - az algoritmus nyomon követi az ideális egyenes és a már kirajzolt pixelpozíció közötti eltérést.
  - Amikor ez az eltérés egy bizonyos határértéket átlép, a rajzolás a másik irányba lép tovább.
  - Így a kiválasztott pixelek sorozata mindig a lehető legjobban követi a matematikai egyenest
    - anélkül, hogy valós számokkal kellene számolni vagy kerekítési műveleteket végezni.

Aliasing - élsimítás...

Aliasing és élsimítás (anti-aliasing)

- Az aliasing jelensége a raszteres megjelenítés egyik alapvető és elkerülhetetlen problémája
- Közvetlenül abból fakad, hogy folytonos geometriai alakzatokat diszkrét képpontok segítségével ábrázolunk
    - a folytonos forma információtartalmának egy része szükségszerűen elveszik.
- **Élsimítás (anti-aliasing):** olyan technikák összessége, amelyek célja az aliasing vizuális hatásainak csökkentése.

- **Alapgondolata:**
    - a pixelek ne kizárólag be- vagy kikapcsolt állapotban szerepeljenek,
    - hanem a részleges fedettség mértékének megfelelő szín- vagy fényerőértéket kapjanak
    - Így az élek átmenetei simábbnak tűnnek, még akkor is, ha a geometriai felbontás változatlan marad

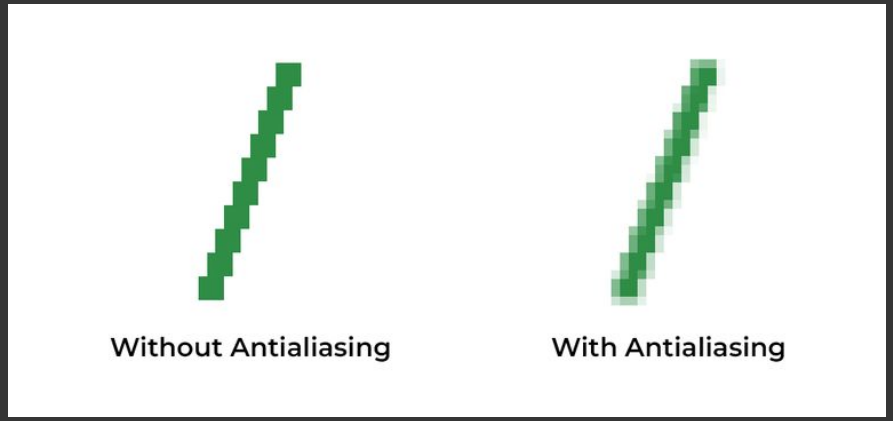
Aliasing és élsimítás (anti-aliasing)

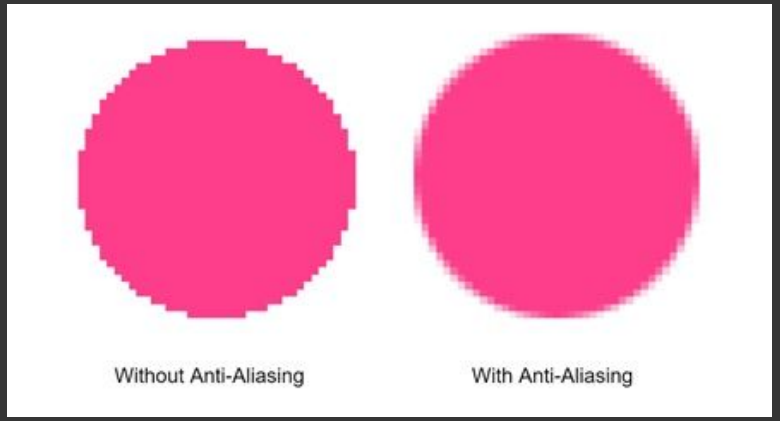
- Az élsimítás elméleti alapja az, hogy egy pixel nem pontszerű mintavételként, hanem egy kis területként értelmezhető
- Ha az ideális vonal csak részben halad át ezen a területen, akkor a pixel színének ennek arányában kellene hozzájárulnia a végső képhez.
- Az anti-aliasing technikák ezt az elvet különböző közelítésekkel valósítják meg, több-kevesebb számítási költséggel.
- A legegyszerűbb élsimítási megoldások a vonalak és élek mentén módosítják a pixelek intenzitását
    - úgy, hogy a szomszédos pixelek fokozatos átmenetet alkossanak.
- Ez a módszer csökkenti a lépcsőzetességet, ugyanakkor enyhe elmosódást eredményezhet.
- A vizuális cél nem a geometriai pontosság növelése, hanem az emberi látás számára kellemesebb kép létrehozása

Aliasing és élsimítás (anti-aliasing)

- **Fontos:** az élsimítás nem tünteti el teljesen az aliasingot, hanem annak észlelhetőségét csökkenti.
- A diszkrét megjelenítés korlátai továbbra is fennállnak,
  - azonban az élek mentén megjelenő nagy kontrasztú ugrások mérséklődnek.
- Ez különösen fontos:
  - nagy felbontású kijelzőkön
  - olyan alkalmazásokban, ahol a vizuális minőség kiemelt szerepet játszik
    - például grafikai tervezésben, játékokban vagy vizualizációkban

Aliasing és élsimítás (anti-aliasing)





Anti-aliasing módszerek

- A legegyszerűbb és legklasszikusabb módszer a **supersampling anti-aliasing** (**SSAA**).
- **Lényege:**
  - minden pixelhez több mintavételi pontot számítunk a képernyőn belül,
  - majd az ezekből származó színértékeket átlagoljuk.
- Ezáltal a pixelek részleges lefedettségét is figyelembe vesszük, ami simább, kevésbé recézett éleket eredményez.
- Az SSAA nagyon hatékony a vizuális minőség növelésére,
  - de számításigényes, mivel minden pixelhez több „virtuális pixelt” kell kiszámítani.

SSA algoritmus

MSAA algoritmus

- A supersampling egy továbbfejlesztett változata a **multisample anti-aliasing (MSAA)**
- Modern játékokban és grafikus motorokban gyakran alkalmaznak
- **Az MSAA lényege:**
  - a geometria élei mentén történő mintavételre koncentrál, így nem minden pixel teljes felületét, csak az élekhez tartozó mintákat számítja.
  - Ez jelentősen csökkenti a számítási igényt az SSAA-hoz képest, miközben az élek mentén is hatékonyan simítja a recézettséget.
  - Az MSAA különösen jól működik 3D jelenetekben, ahol sok élszakasz fut párhuzamosan, és a pixelek belseje viszonylag homogén.

MSAA algoritmus

TAA algoritmus

- A játékok és modern grafikus alkalmazások további, fejlettebb megoldásai között megtalálható a **temporal anti-aliasing (TAA)**.
- A TAA a képkockák közötti mozgást is figyelembe veszi, így nemcsak a statikus, hanem a mozgó élek recézettségét is csökkenti.
- Ez a módszer az előző képkockák mintáit kombinálja az aktuális képkocka pixeleivel,
    - és finom átmeneteket hoz létre, ami a gyorsan mozgó objektumoknál is sima képet biztosít.
- A TAA jelentős előnye:
    - alacsonyabb számítási igénnyel érhető el, mint a nagyfelbontású supersampling, ugyanakkor jól kezeli a mozgó jeleneteket is.

FXAA algoritmus

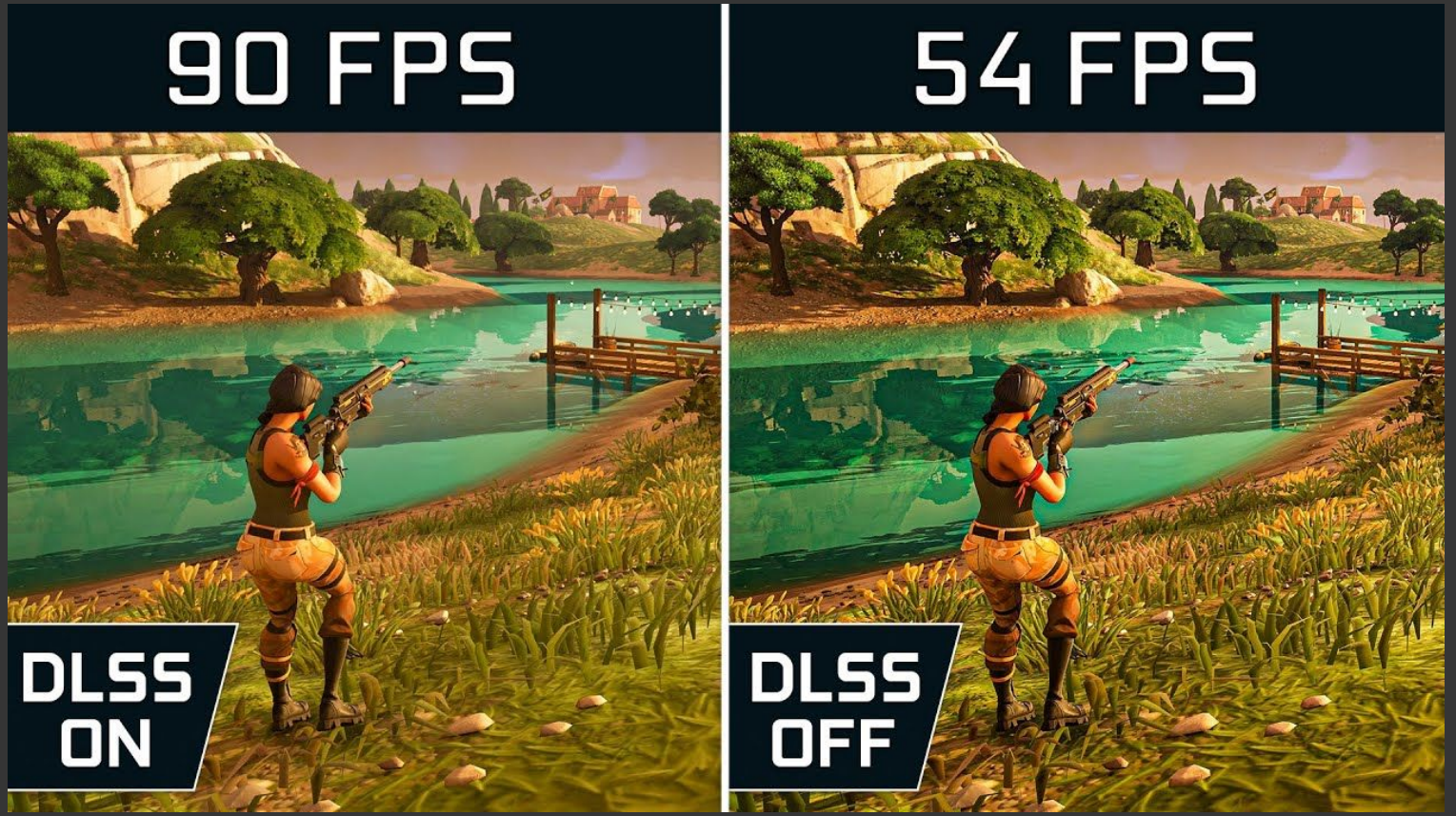
- A legmodernebb játékokban megjelennek a post-processing anti-aliasing technikák: pl. **FXAA (Fast Approximate Anti-Aliasing)**
- Az FXAA nem a pixel geometriai mintavételén alapul, hanem a renderelt kép pixelei közötti kontrasztot vizsgálja
    - és a magas kontrasztú, recézett élek mentén finom simítást alkalmaz.
- Ez a módszer rendkívül gyors, és szinte minden grafikus hardveren működik
- Hátránya, hogy a kép enyhén elmosódhat, és a nagyon részletes textúráknál csökkentheti az élességet.

FXAA algoritmus

DLSS algoritmus

- A legújabb trendek a deep learning alapú anti-aliasing módszerek felé mutatnak
  - Pl. például az **NVIDIA DLSS** (Deep Learning Super Sampling) vagy a AMD **FSR** (FidelityFX Super Resolution) 2.0/3.0.
- Ezek a megoldások mesterséges intelligenciát és neurális hálókat használnak a kép feljavítására,
  - a recézett élek simítására és a képkockák felskálázására.
- **DLSS:** alacsonyabb felbontású képet renderel, majd neurális hálóval rekonstruálja a végső, magas felbontású képet simított élekkel és részletgazdag textúrákkal.
- Az AI-alapú anti-aliasing jelentősen javítja a teljesítményt, mivel a grafikus hardver kevesebb pixelt számol ki közvetlenül, miközben a vizuális minőség magas marad.

DLSS ON vs OFF



Képfeldolgozó programok...

Képfeldolgozó programok

- A képfeldolgozó programok és eszközök célja:
  - digitális képek módosítása, javítása, elemzése vagy értelmezése különböző algoritmusok segítségével.
- A képadatok tartalmára koncentrálnak, és a képet nem vizuális objektumként, hanem numerikus adathalmazként kezelik.
- Egy digitális kép ebben a megközelítésben pixelek rendezett mátrixa, ahol minden pixelhez egy vagy több számérték tartozik.
- A legismertebb képfeldolgozó programok közé tartoznak az általános célú képszerkesztők, mint például az **Adobe Photoshop** vagy a **GIMP**.
- Ezek a programok elsősorban vizuális felhasználásra készültek, azonban működésük alapját klasszikus képfeldolgozási algoritmusok adják.

Képfeldolgozó programok

- Ezek a programok gyakran lehetőséget biztosítanak különböző bitmélységű munkaterületek használatára is.
- Ez különösen fontos a professzionális képfeldolgozás és **HDR** tartalmak esetében,
  - mivel a nagyobb bitmélység csökkenti az információvesztést a feldolgozás során.
- A felhasználó számára ez sokszor csak annyiban jelenik meg, hogy egy kép „jobban bírja” a szerkesztést,
  - de a háttérben komoly numerikus pontosságbeli különbségek állnak

Képfeldolgozó programok

- A képfeldolgozás másik fontos területét a tudományos és mérnöki célú eszközök képviselik.
- Ezek közé tartoznak például az **ImageJ** vagy a **MATLAB**-alapú képfeldolgozó környezetek.
- Ezeknél az eszközöknél a hangsúly nem az esztétikai megjelenésen, hanem az objektív mérésen és elemzésen van.
- A képek gyakran mérési adatok vizuális reprezentációi, például orvosi felvételek, mikroszkópos képek vagy műholdas adatok.
- Ebben a környezetben a képfeldolgozás célja sokszor nem a kép „szebbé tétele”, hanem az információ kinyerése
    - például objektumok felismerése, élek detektálása vagy statisztikai jellemzők számítása.

Képfeldolgozó programok

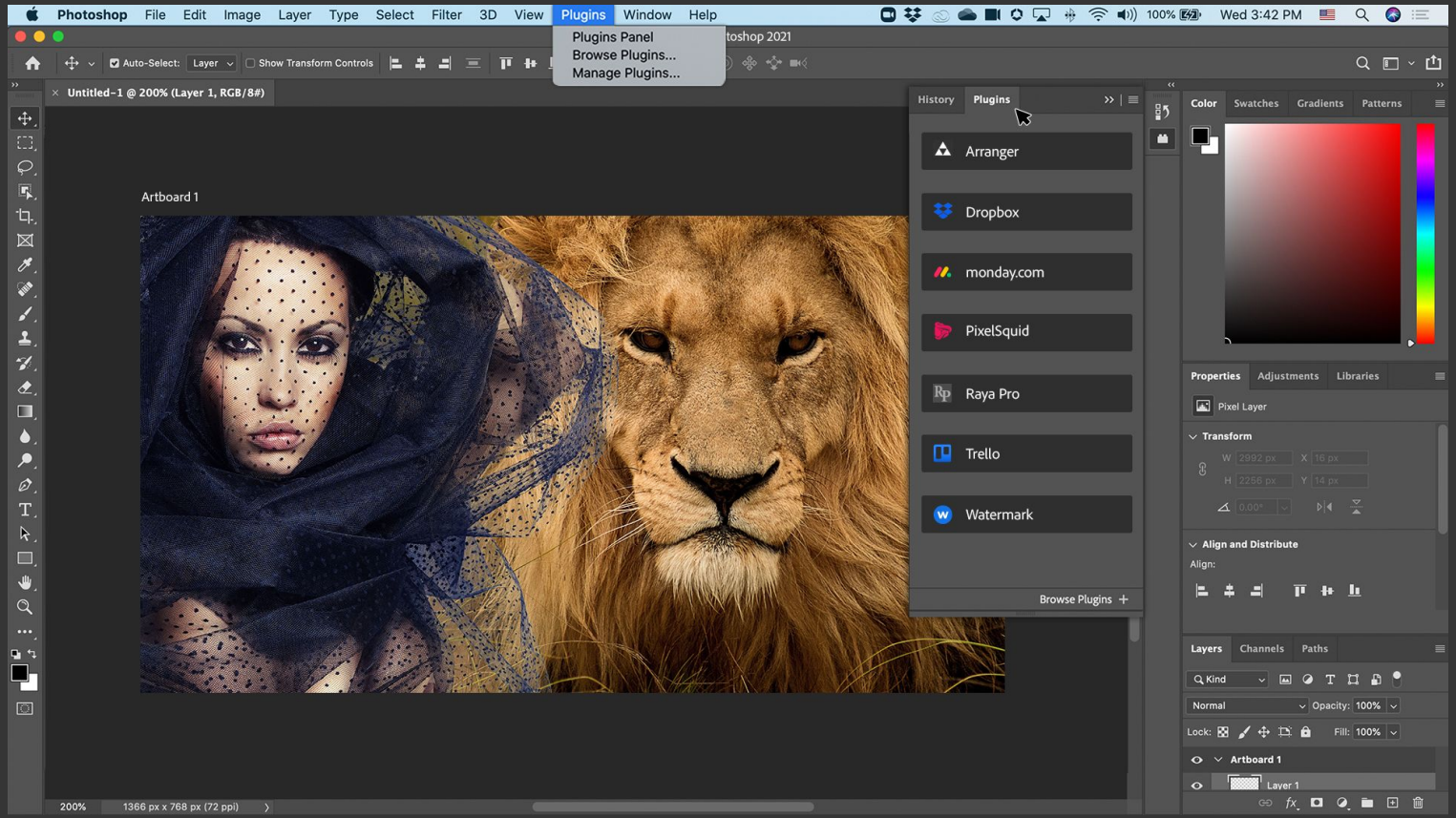
- A modern képfeldolgozásban egyre nagyobb szerepet kapnak a programozható könyvtárak és keretrendszerek.
- Ilyen eszköz például az **OpenCV**, amely széles körű funkcionalitást kínál képek betöltésére, feldolgozására és elemzésére.
- Ezek az eszközök már nem grafikus felhasználói felületre épülnek, hanem közvetlenül programkódból érhetők el.
- A képfeldolgozási műveletek itt algoritmusok formájában jelennek meg, amelyek explicit módon manipulálják a pixelértékeket.

Ismertebb szoftverek

Adobe Photoshop

- A legismertebb és legelterjedtebb professzionális képszerkesztő szoftver,
- A képfeldolgozás és a digitális grafika határterületén helyezkedik el.
- Kiváló példa arra, hogyan jelennek meg klasszikus képfeldolgozási műveletek felhasználóbarát formában

Ismertebb szoftverek

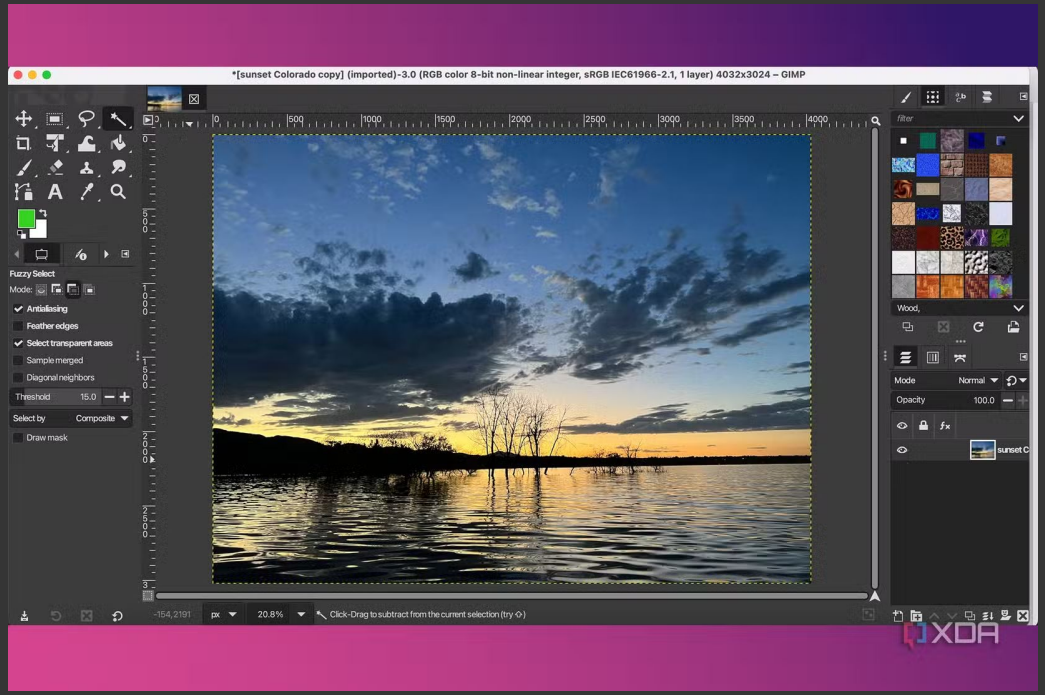


Ismertebb szoftverek

GIMP (GNU Image Manipulation Program)

- Nyílt forráskódú alternatívája a Photoshopnak, amely széles körű képfeldolgozási és képszerkesztési funkciókat kínál.
- Oktatási környezetben különösen előnyös, mivel szabadon hozzáférhető és jól demonstrálja az alapvető algoritmusokat.

GIMP 3.0


*[sunset Colorado copy] (imported)-3.0 (RGB color 8-bit non-linear integer, sRGB IEC61966-2.1, 1 layer) 4032x3024 – GIMP
Fuzzy Select
Mode:
Antialiasing
Feather edges
Select transparent areas
Sample merged
Diagonal neighbors
Threshold 15.0
Select by Composite
Draw mask
Wood,
Mode Normal
Opacity 100.0
-154,2191 px 20.8% Click-Drag to subtract from the current selection (try Ctrl)

Ismertebb szoftverek

Aseprite

- Az Aseprite kifejezetten pixelgrafikára és sprite-alapú grafikai tartalmak készítésére specializált eszköz.
- Oktatási szempontból különösen értékes, mert a pixelek szerepe itt közvetlenül és kézzelfogható módon jelenik meg.
- A program jól demonstrálja az alacsony felbontás, a színpaletták használata, az aliasing és az anti-aliasing kérdését, valamint az animáció alapjait.

Aseprite


Edit Sprite Layer Frame Select View Help
Fast Rotation
Preview
Palette Editor (F4)
Entry: 5
Background

Ismertebb szoftverek

Microsoft Paint

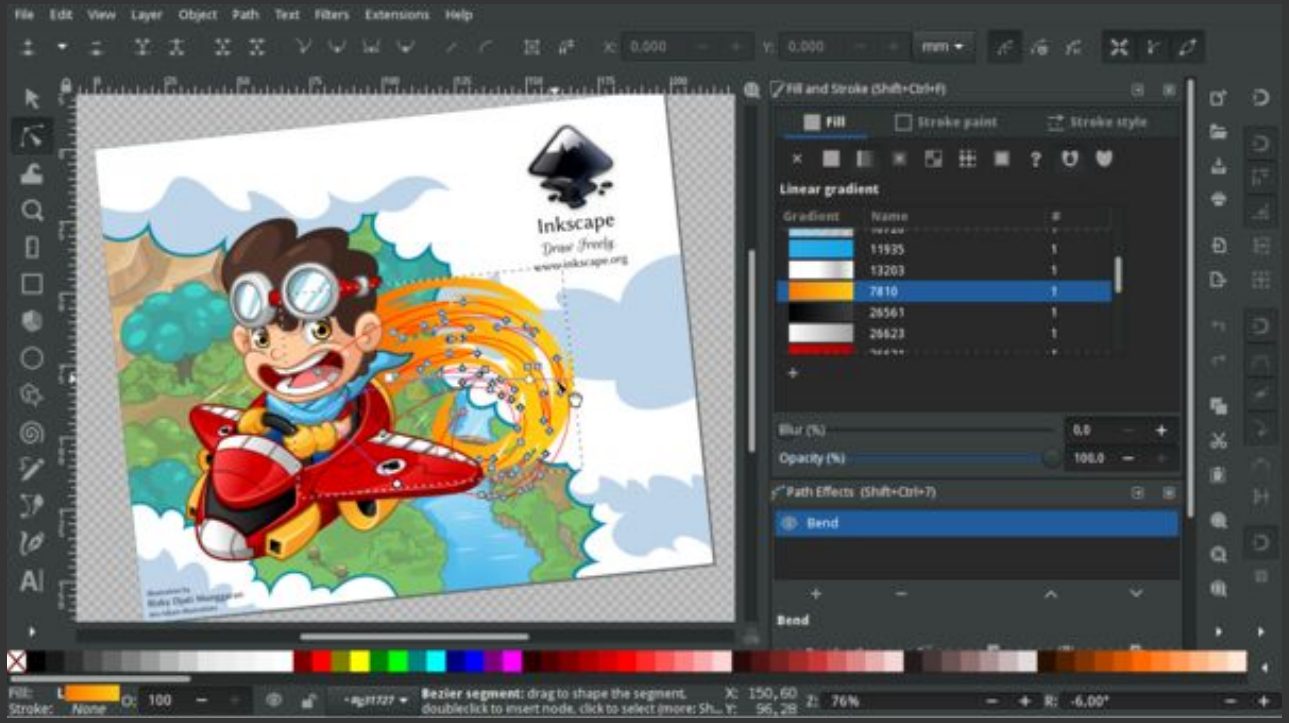
- A Paint egy rendkívül egyszerű raszteres rajzolóprogram,
- Bár funkcionalitásában korlátozott, jól használható az alapvető dolgokra.
- A pixel szintű rajzolás, a felbontás, a színmélység és az egyszerű képfeldolgozási műveletek itt minden absztrakció nélkül jelennek meg.
- A Paint jó kiindulópont lehet a „mi is az a digitális kép” kérdés gyakorlati megértéséhez.

Ismertebb szoftverek

Inkscape

- Az Inkscape nyílt forráskódú vektorgrafikus szerkesztőprogram
- Kiválóan alkalmas a vektoros és raszteres grafika közötti különbségek bemutatására.
- Az objektumalapú szerkesztés, a Bézier-görbék és a felbontásfüggetlen ábrázolás.
- Az Inkscape különösen hasznos akkor, amikor a vektorgrafika raszteresítéséről és a megjelenítés során fellépő aliasing jelenségről esik szó.

Inkspace


File Edit View Layer Object Path Text Filters Extensions Help
Fill and Stroke (Shift+Ctrl+F)
Fill
Stroke paint
Stroke style
Linear gradient
Gradient
Name
11935
13203
7810
26561
26623
Blur (%)
Opacity (%)
Path Effects (Shift+Ctrl+7)
Bend
Inkscape
Draw Freely
Bezier segment: drag to shape the segment, doubleclick to insert node, click to select (more: Sh...

Ismertebb szoftverek

Spriter

- A Spriter egy animációkészítő eszköz, amely elsősorban 2D játékokhoz használatos.
- Működése eltér a hagyományos képkocka-alapú animációtól
- Előre elkészített raszteres elemeket, úgynevezett sprite-okat mozgat és transzformál.
    - Pl. csontváz alapú animáció

Spriter

Köszönöm a figyelmet!