Zachodniopomorski
Uniwersytet
Technologiczny
w Szczecinie

http://bazyluk.net/dydaktyka

Bartosz Bazyluk

Wyświetlanie obrazu

Techniki wyświetlania obrazu komputerowego.

Wydział
Informatyki

Grafika Komputerowa i Wizualizacja, Informatyka S1, II Rok

WYŚWIETLANIE OBRAZU

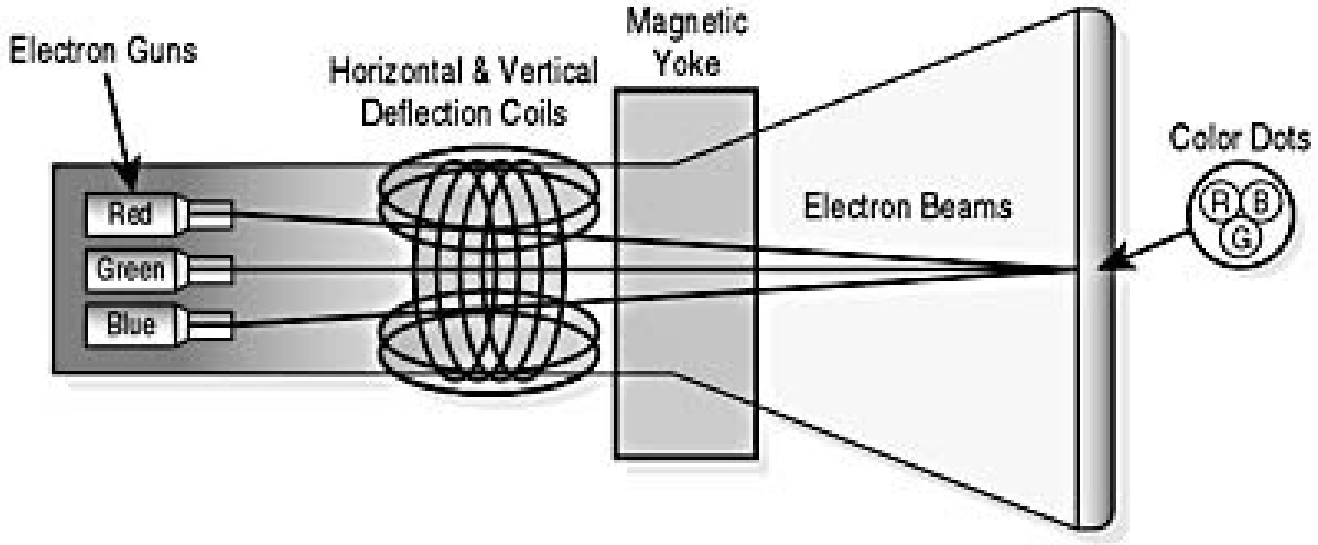
- Techniki wyświetlania obrazu
  - Wyświetlacze **CRT**
  - Wyświetlacze **LCD**
  - Wyświetlacze **PDP**
  - Wyświetlacze **OLED**
  - Projektory **DLP**
  - Inne: wyświetlacze **laserowe**, wyświetlacze **LED**, ...



Źródła obrazów:
http://trustedreviews.com/
http://accad.osu.edu/~elaine/
http://phonearena.com/

WYŚWIETLANIE OBRAZU

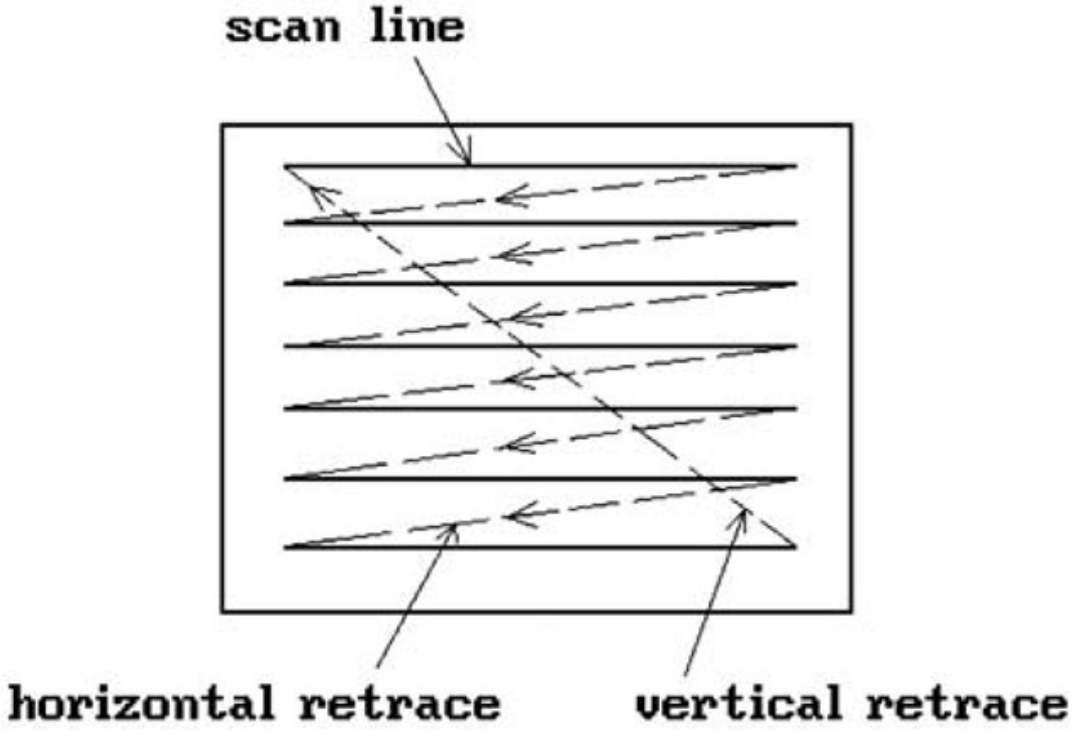
- Wyświetlacze **CRT** (kineskopowe)
  - Działa elektronowe emitują **wiązki elektronów**, które pobudzają warstwę luminoforu
  - Pobudzony luminofor **emituje światło**
  - Cewki elektryczne **odchylają** wiązkę elektronów, by trafiała w różne części ekranu



Źródła obrazów:
http://brainbell.com/
http://obsoletetellyemuseum.blogspot.com/

WYŚWIETLANIE OBRAZU

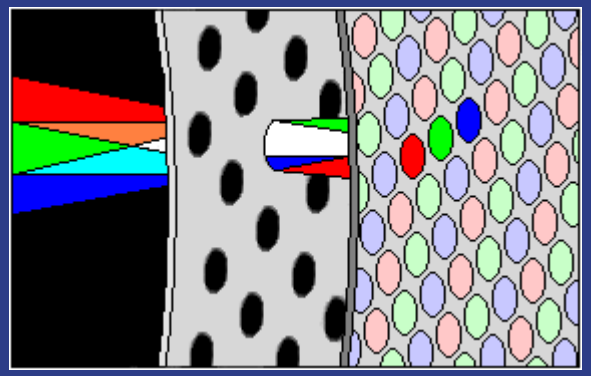
- Wyświetlacze **CRT** (kineskopowe)
  - Sposób odświeżania całego **rastra**: **odchylanie pionowe** (m. cz.) służące do przechodzenia kolejnych wierszy i **odchylanie poziomie** (w. cz.) do przechodzenia kolumn w obrębie każdego z wierszy

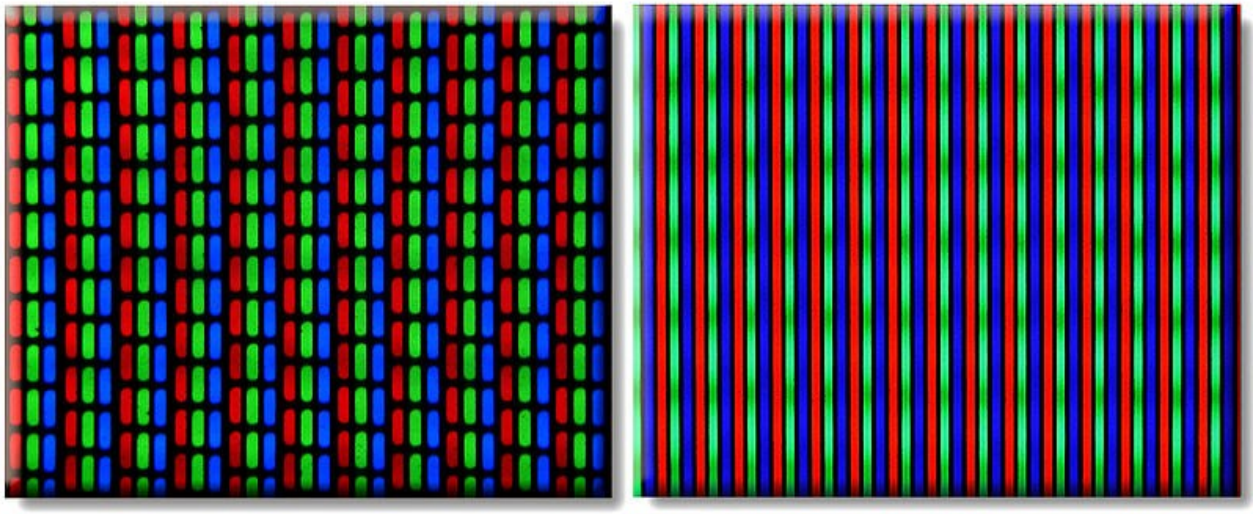


Źródła obrazów:
http://what-when-how.com/
Wikipedia

WYŚWIETLANIE OBRAZU

- Wyświetlacze **CRT** (kineskopowe)
  - Kolorowy obraz powstaje dzięki użyciu rodzajów **luminoforu emitującego różnokolorowe światło**, najczęściej RGB
  - Wiązka elektronów trafia w odpowiednią komórkę z luminoforem po przejściu przez perforowaną **maskę**

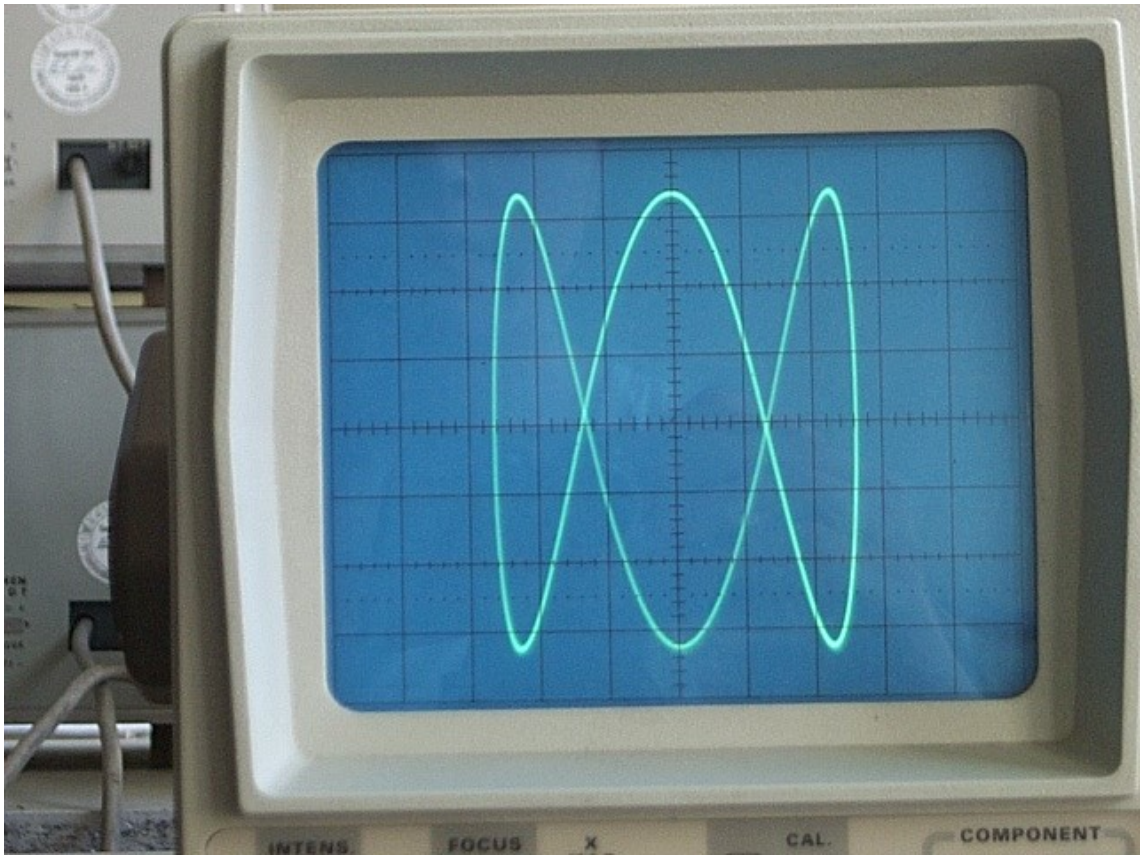




Źródła obrazów:
http://sarcasmeur.free.fr/
http://ttlg.com/

WYŚWIETLANIE OBRAZU

- Wyświetlacze **CRT** (kineskopowe)
  - Szczególny przypadek: **oscyloskop**
    - Strumień elektronów jest sterowany wyświetlanym sygnałem, zamiast przechodzenia całego rastra przy pomocy cewek odchylania pionowego i poziomego



Źródło obrazu:
Wikipedia

WYŚWIETLANIE OBRAZU

- Wyświetlacze **CRT** (kineskopowe)

Zalety ekranów CRT:

Dobra **reprodukcja kolorów**
Duży **kontrast**
Dobre **kąty widzenia**
Jakość niezależna od **rozdzielczości**
Bardzo duża **szybkość**

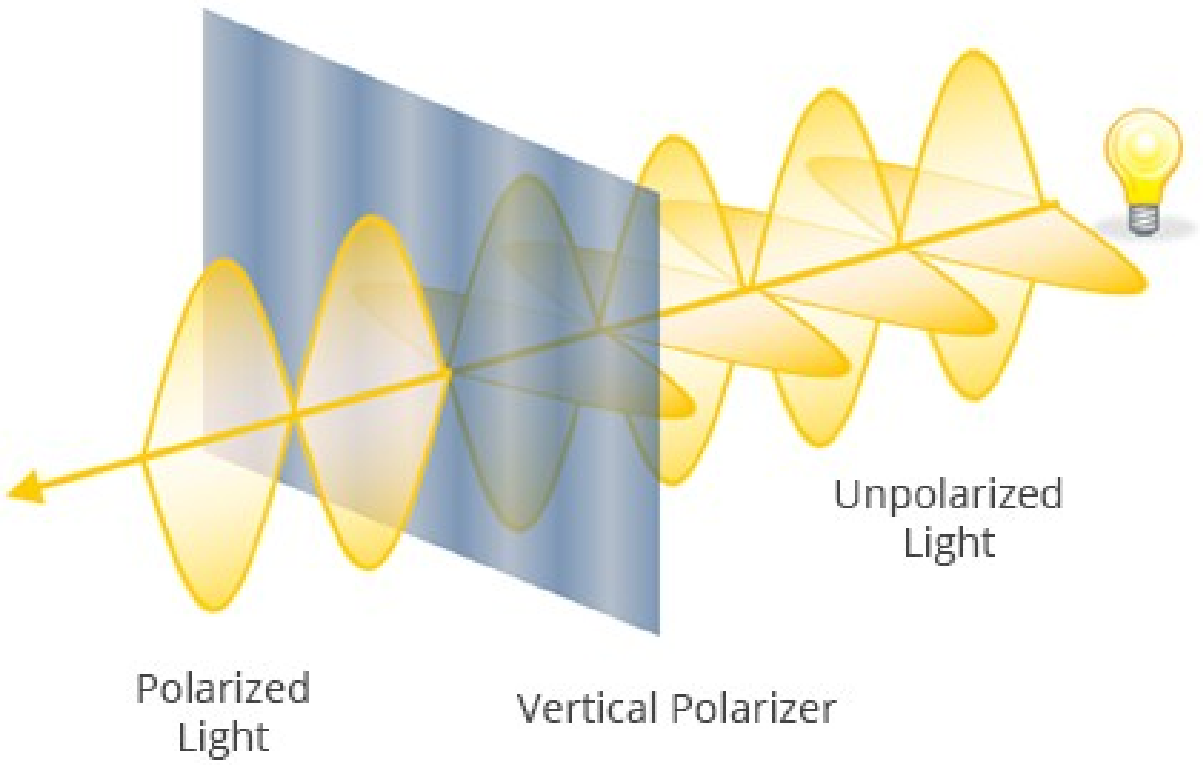
Główne **wady**:

Gabaryty
Migotanie obrazu
Zużycie **energii**
Problemy z **ostrością obrazu**
Szkodliwość dla zdrowia

Źródło obrazu:
http://www.jestineyong.com

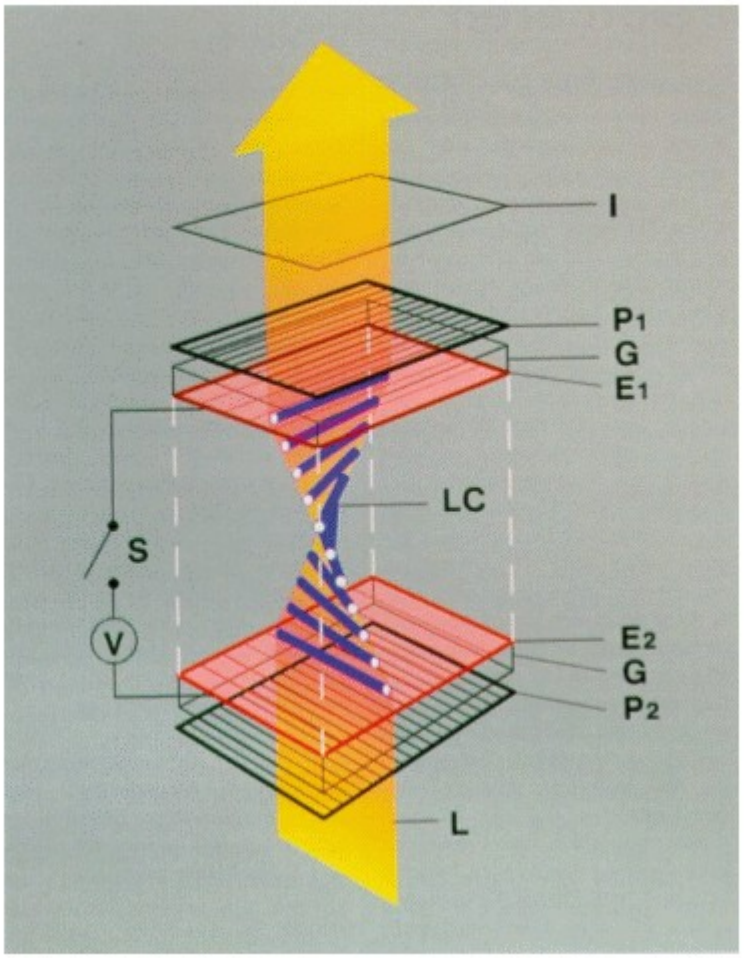
WYŚWIETLANIE OBRAZU

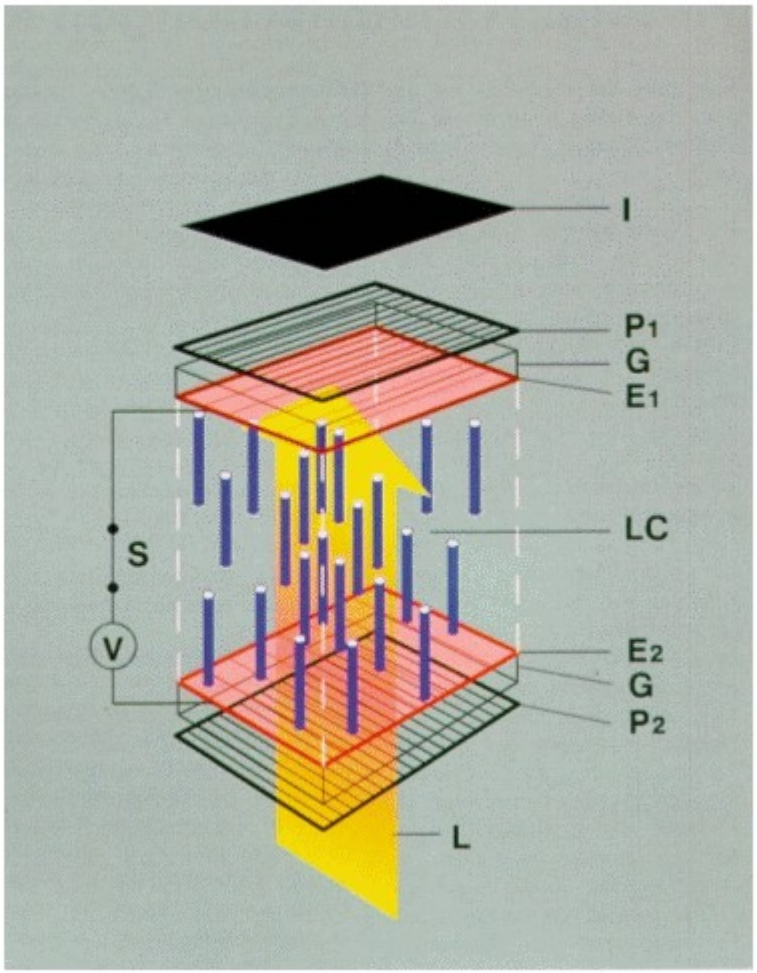
- Liniowa **polaryzacja** światła
  - Światło jest **falą poprzeczną**
  - W fali **niespolaryzowanej** oscylacje odbywają się z jednakową amplitudą **w różnych kierunkach prostopadłych do kierunku rozchodzenia się fali**
  - Po przejściu przez polaryzator oscylacje odbywają się tylko **w jednej płaszczyźnie**



Źródło obrazu:
http://bigshotcamera.com/

- **Ciekłe kryształy** (ang. *Liquid Crystals*)
  - Warstwa ciekłokrystaliczna po przyłożeniu napięcia **skręca płaszczyznę polaryzacji**

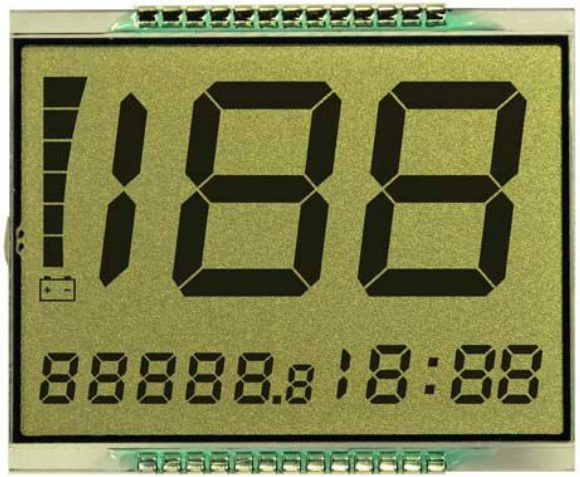




Źródło obrazu:
http://www.wikipedia.org

WYŚWIETLANIE OBRAZU

- Działanie panelu **LCD** (ang. *Liquid Crystal Display*)
  - **Matryca pasywna** – schemat wyświetlanego obrazu nie zależy od stanu elektrycznego



Źródło obrazu:
http://www.directindustry.com

WYŚWIETLANIE OBRAZU

- Działanie panelu **LCD** (ang. *Liquid Crystal Display*)

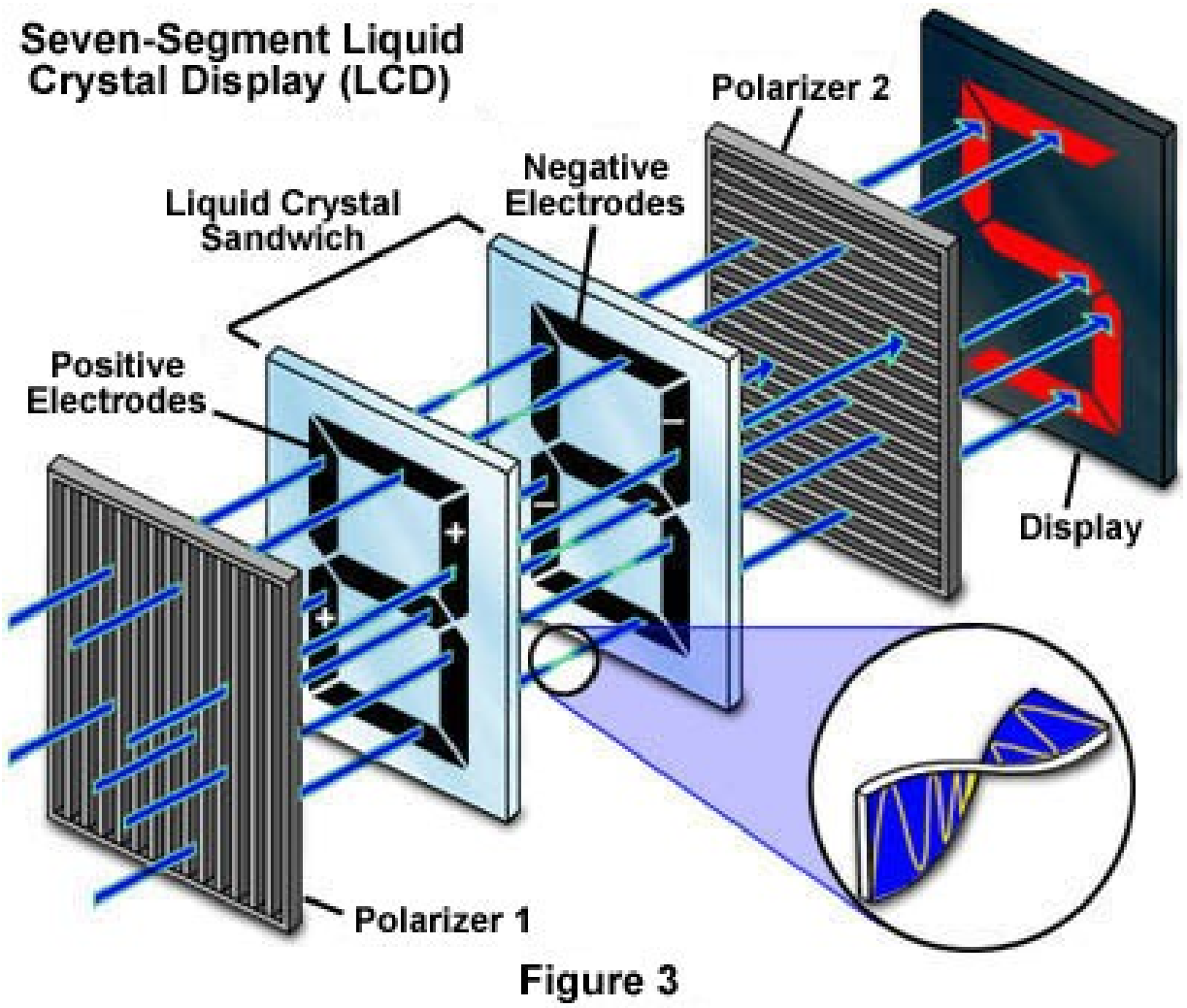


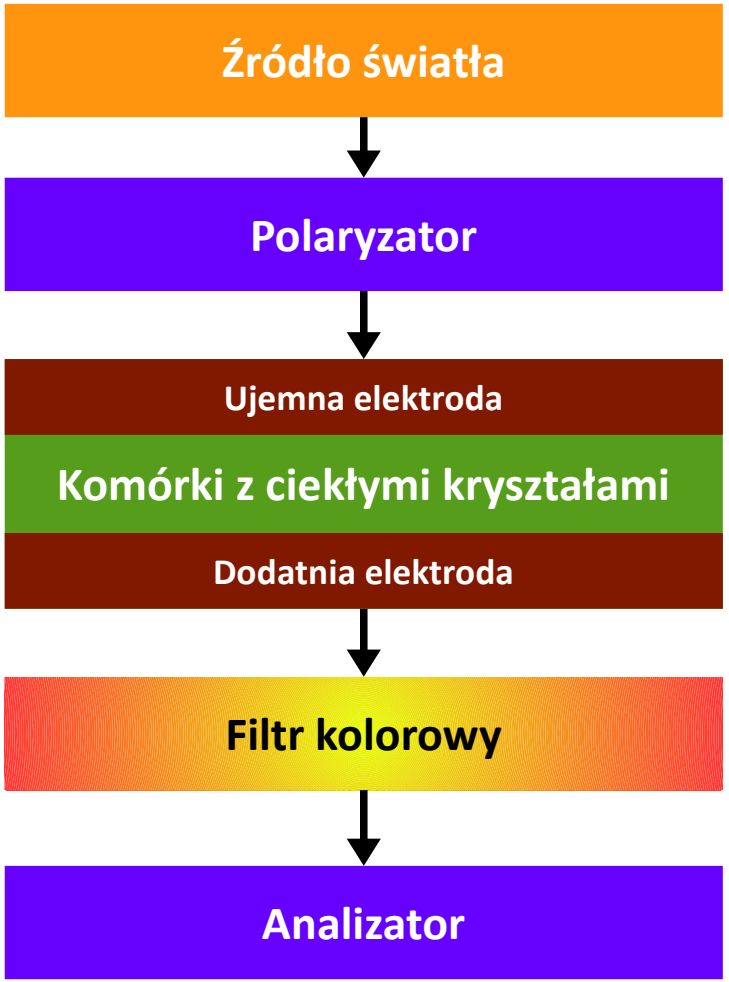
Figure 3

Źródło obrazu:
http://www.olympusmicro.com

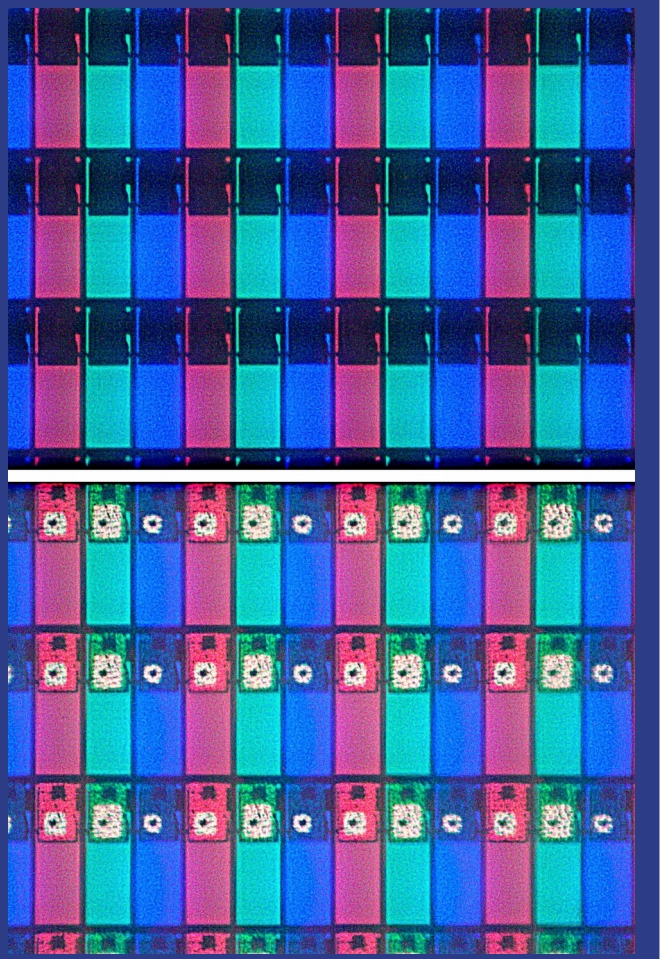
WYŚWIETLANIE OBRAZU

Źródło obrazu:
http://www.tradekorea.com

- Działanie **kolorowego** panelu LCD
  - **Matryca pasywna** – schemat wyświetlanego obrazu nie zależy od stanu elektrycznego

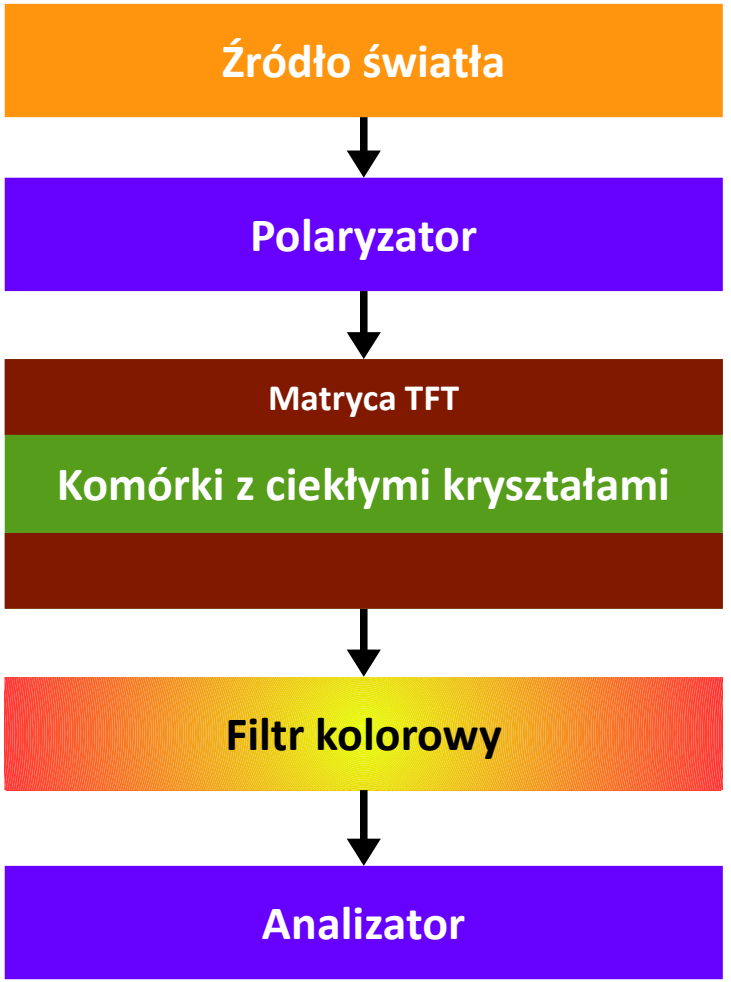


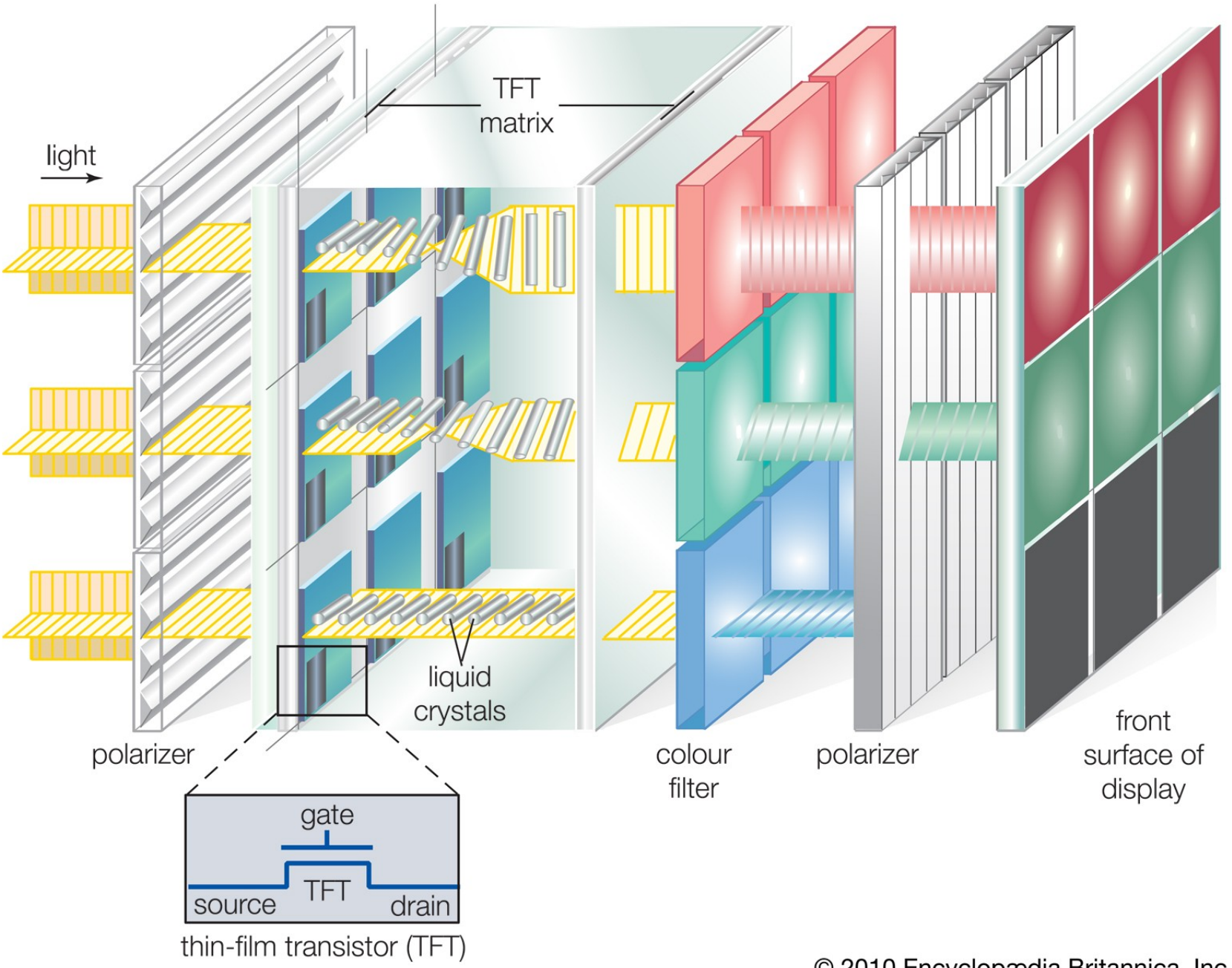
WYŚWIETLANIE OBRAZU



Źródło obrazu:
http://www.wikipedia.org

- Działanie panelu **AMLCD** (ang. *Active Matrix LCD*)
  - Komórkami LC steruje **aktywna matryca TFT** (ang. *Thin Film Transistor*)





WYŚWIETLANIE OBRAZU

Ogólne **zalety** ekranów LCD:

Małe **gabaryty**
Brak migotania obrazu
Ostrość *(w rozdzielczości „natywnej”)*

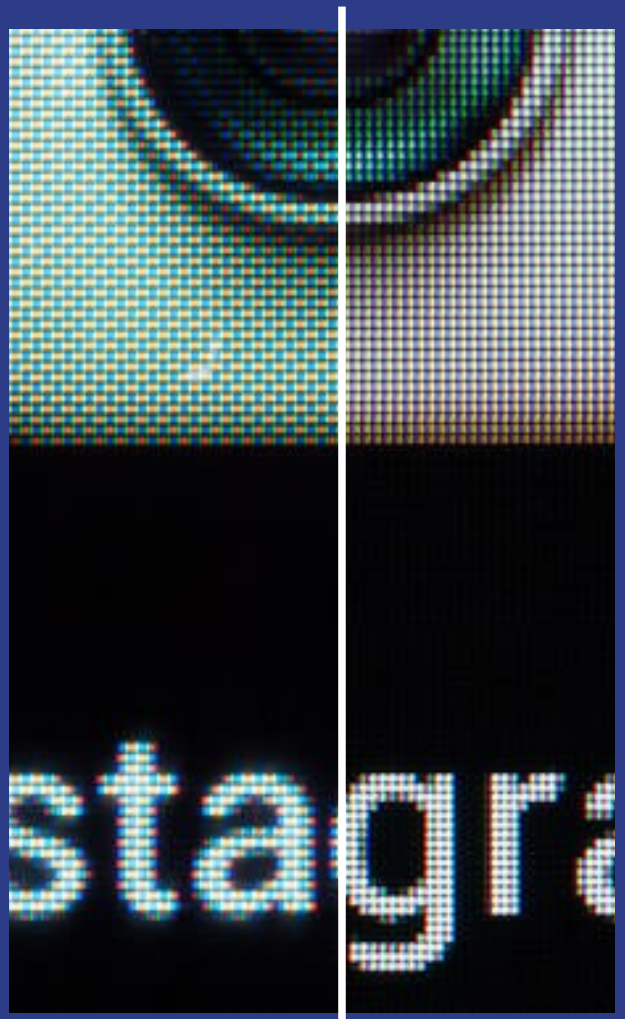
Główne **wady**:

Ograniczone **kąty widzenia**
Wymagane **podświetlenie**
Świecąca **czerń**

Źródło obrazu:
http://www.flatscreenexpert.com

- Technologie budowy i działania paneli **AMLCD**
  - Panele **TN** (ang. *twisted nematic*)
    - **Tanie**, **szybkie**, **małe kąty widzenia**
  - Panele **IPS** (ang. *in-plane switching*)
    - *S-IPS*, *E-IPS*, **-IPS*...
    - Dwa tranzystory na każdy piksel, **odwzorowanie kolorów** **dobre kąty widzenia**, większe straty światła, **powolne**, **droższe**
  - Panele **VA** (ang. *vertical alignment*)
    - *MVA*, *PVA*, **VA*...
    - **Większy kontrast**, **dobre kąty widzenia**, **powolne**

WYŚWIETLANIE OBRAZU



AMOLED AMLCD

Źródło obrazu:
http://www.stuff-review.com

- Działanie panelu **AMOLED** (ang. *Active Matrix Organic LED)*
  - Zbudowane w oparciu o warstwę emisyjną zbudowaną z **komórek LED**
  - LEDy złożone z **polimerów przewodzących**
  - Stanem komórek steruje **aktywna matryca TFT**

WYŚWIETLANIE OBRAZU

- Działanie panelu **AMOLED** (ang. *Active Matrix Organic LED)*

Zalety względem LCD:

Kąty widzenia
Jasność i **kontrast**
Odwzorowanie **barw**
Giętkie wyświetlacze
Prosta budowa
Niski **koszt**

Główne **wady**:

Pobór prądu dla jasnych obrazów
Ograniczenia **patentowe**

Źródło obrazu:
http://www.digitalversus.com

WYŚWIETLANIE OBRAZU

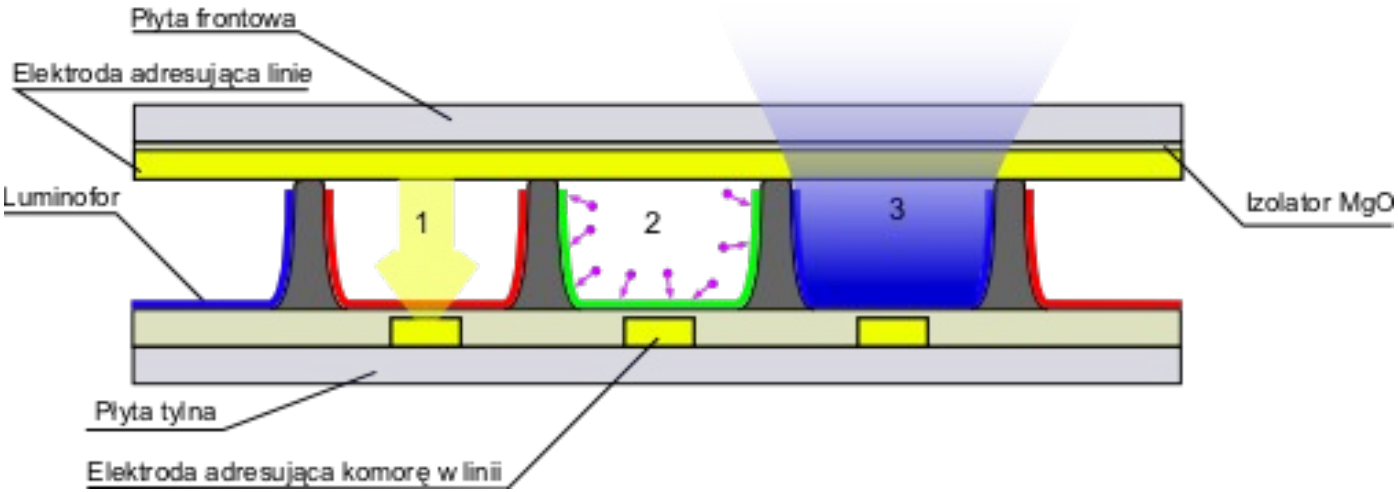
Zalety względem LCD:

Wyższy **kontrast**
Głębsza **czerń**
Lepsze **kąty widzenia**
Duże **rozmiary** ekranu

Główne **wady**:

Pobór **prądu**
Wydzielanie **ciepła**
Wypalanie luminoforu
Migotanie obrazu

- Działanie panelu **PDP** (plazmowego)
  - Przyłożenie napięcia do komórek zawierających **mieszaninę gazów** powoduje doprowadzenie ich do **stanu plazmy**
  - Emitują wówczas **światło ultrafioletowe**
  - Pobudzany nim **luminofor** emituje światło widzialne
  - Zróżnicowanie poziomów jasności uzyskuje się poprzez **modulację impulsową** PWM (ang. *pulse-width modulation*)



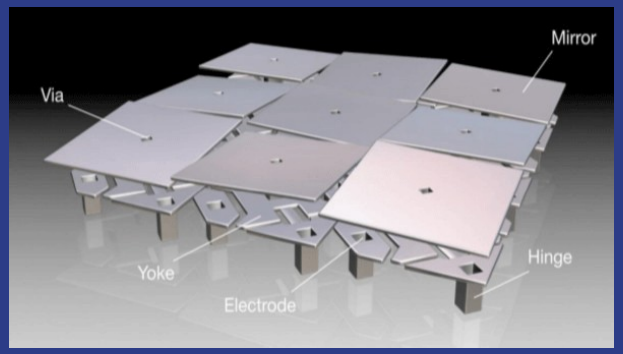
Źródło obrazu:
Wikipedia

WYŚWIETLANIE OBRAZU

- Działanie panelu **PDP** (plazmowego)
  - Problem **wypalania się** luminoforu
    - Wynik działania **wysokiej temperatury** przez długi czas
    - Powoduje powstanie **obrazu-ducha** (ang. *ghost image*)
    - Aby go uniknąć, stosuje się techniki **mikroprzesunięć obrazu** (ang. *pixel orbiting*)

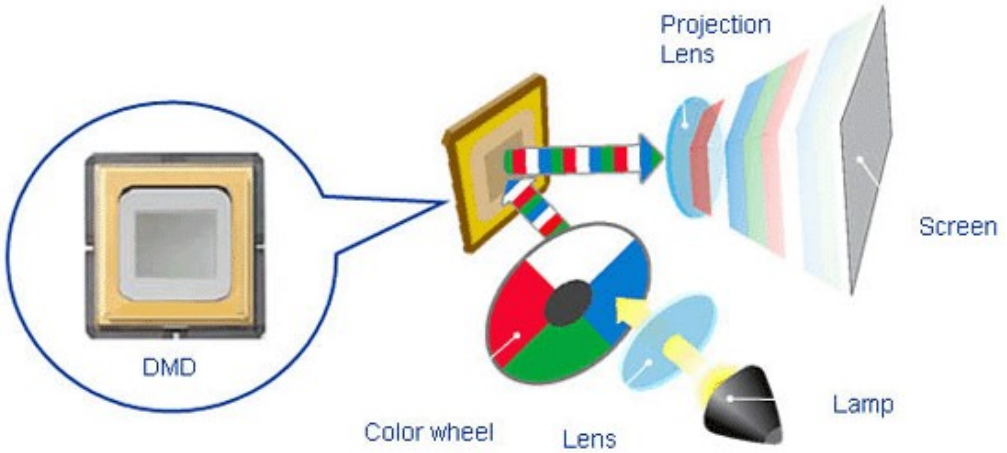
Źródło obrazu:
Wikipedia

WYŚWIETLANIE OBRAZU



Wycinek matrycy *DMD*
złożonej z mikroluster

- Działanie projektora **DLP** (ang. *Digital Light Processing*)
  - Światło emitowane przez silne, wymienne źródło światła, jest odbijane przez **matrycę luster DMD** (ang. *Digital Micromirror Device*) w kierunku obiektywu
    - Każdemu **pikselowi** odpowiada jedno, **ruchome lustro**
    - Zależnie od położenia lustra, światło jest albo kierowane w stronę obiektywu (jasny piksel), albo na radiator (ciemny)
  - Światło może uprzednio przejść przez obracający się **filtr barwny**, kolejno barwiący światło na kolory R, G, B
    - Niektóre projektory wykrozystują inne technologie, np. matryce LCD, trójkolorowe diody LED itp.
  - Po przejściu przez **obiektyw** światło jest **rzucane na ekran**, tworząc na nim finalny obraz

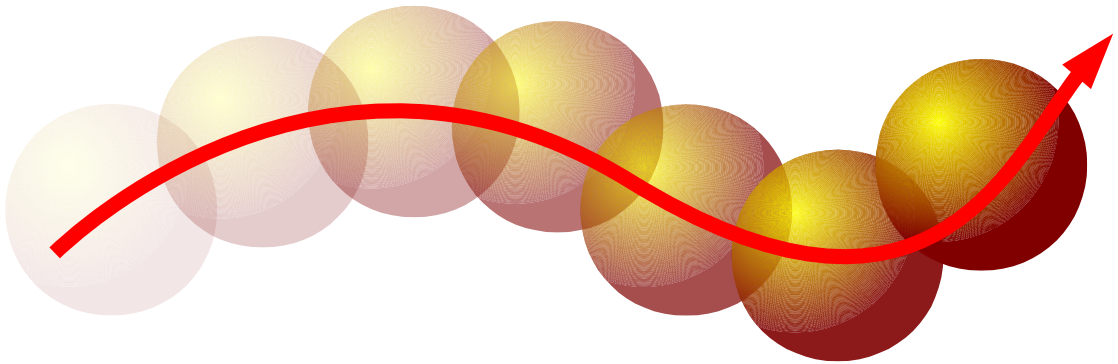


Źródła obrazów:
http://electronicdesign.com
http://kubevent.fr

WYŚWIETLANIE OBRAZU

- Działanie projektora **DLP**
  - Efekt **tęczy**
    - **Artefakt** związany ze sposobem uzyskiwania koloru, opartym na mechanicznie poruszającym się filtrze barwnym
    - Oko poruszając się płynnie podczas śledzenia ruchomego obrazu, zauważa **stany pośrednie** złożone ze składowych kolorystycznych

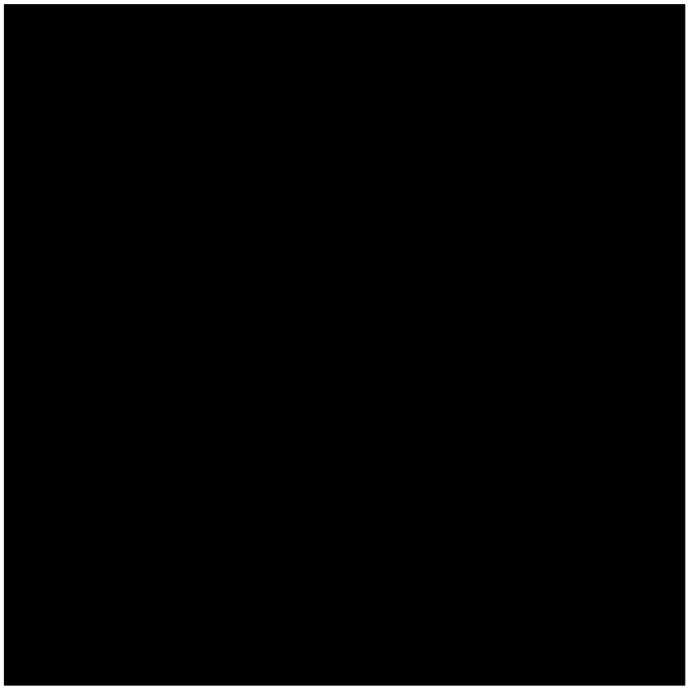
Przy szybkim ruchu wyświetlanej kuli, śledzące ją oko zauważy pośrednie klatki wyświetlane tylko w jednym z kolorów: R, G, B.



Źródło obrazu:
Wikipedia

WYŚWIETLANIE OBRAZU

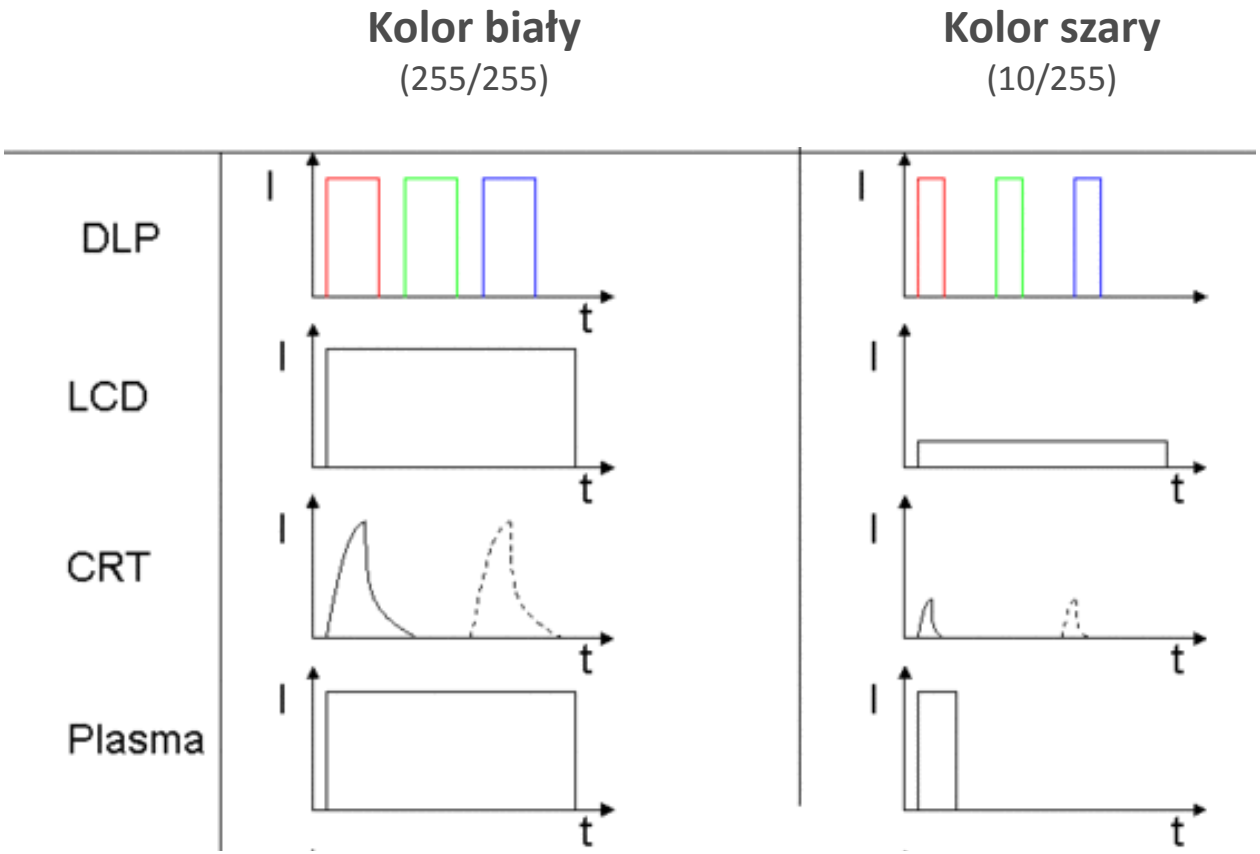
- Działanie projektora **DLP**
  - Efekt **tęczy**



Źródło obrazu:
Wikipedia

WYŚWIETLANIE OBRAZU

- **Różnice** w sposobie wyświetlania klatki koloru białego w czasie, korzystając z różnych technik



Źródło obrazu:
Wikipedia

WYŚWIETLANIE OBRAZU

- Współczesne wyświetlacze komputerowe traktują obraz jako **raster**
- Wyświetlacze działają ze stałą częstotliwością odświeżania (dawniej *odchylania pionowego*)
  - Najczęściej spotykaną częstotliwością odświeżania jest obecnie **60Hz**
  - Dla obrazu **stereoskopowego**, będzie to dwa razy więcej

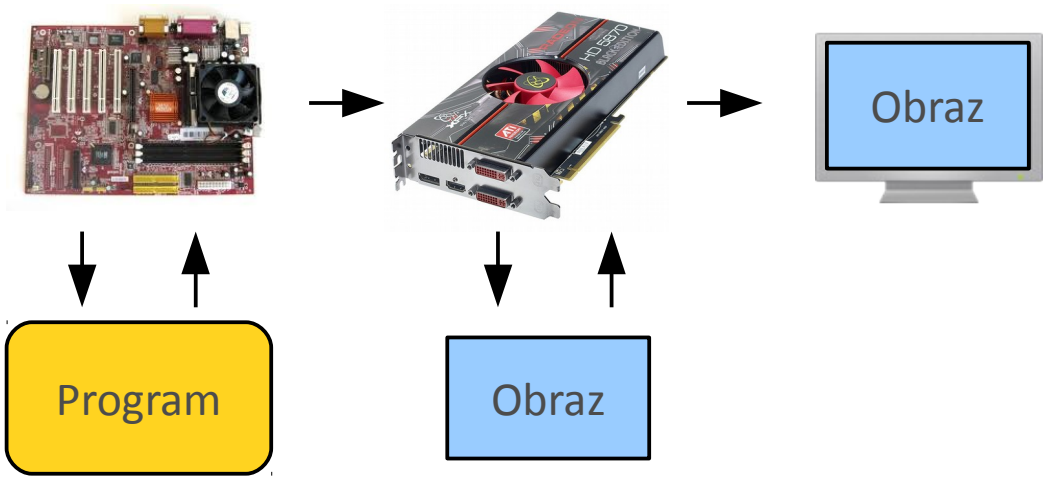
Progresywne wyświetlanie klatki:

1/60s

Źródło obrazu:
http://www.wikipedia.org

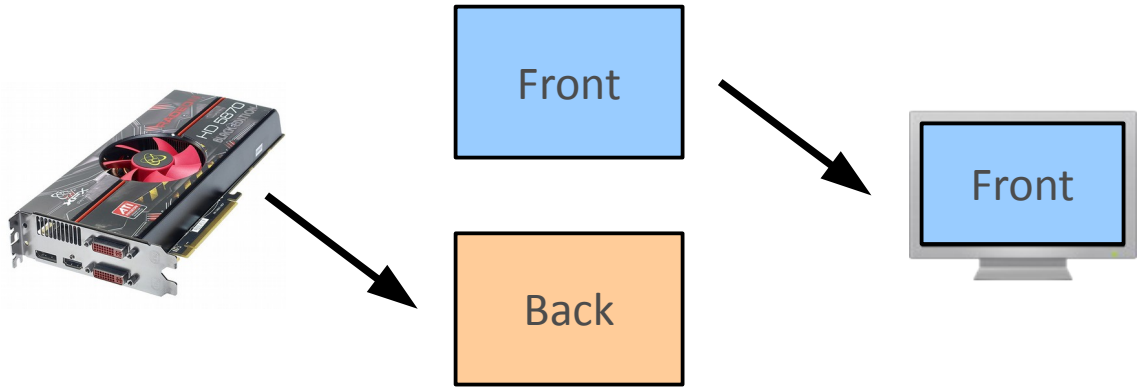
WYŚWIETLANIE OBRAZU

- **Karta graficzna** odpowiada za stworzenie zawartości obrazu monitora, przechowywanie jej i wysłanie do monitora gdy ten będzie gotowy narysować kolejną klatkę
- Obraz do wyświetlenia przechowywany jest w **buforze klatki** (ang. *framebuffer*) w pamięci graficznej (współcześnie np. *GDDR*)
  - Bufor klatki posiada takie same cechy, jak zwyczajny obraz: **wymiary, liczbę kanałów i rozmiar piksela**



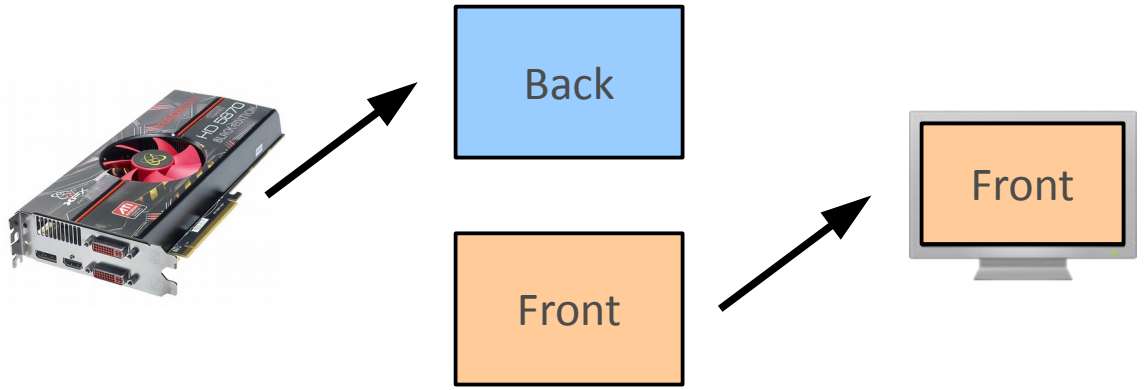
WYŚWIETLANIE OBRAZU

- Aby uniknąć migotania obrazu podczas jego przerysowywania gdy występuje konieczność jego akutalizacji, stosuje się technikę **podwójnego buforowania** (ang. *double buffering*)
- Mamy wówczas dwa bufory klatki:
  - **Bufor "przedni"** (ang. *front buffer*)
    - Jest wyświetlany
  - **Bufor "tylny"** (ang. *back buffer*)
    - Jest aktualizowany
- Gdy zakończy się aktualizacja back buffera, **zamieniane są one miejscami** (ang. *buffer swap*)



WYŚWIETLANIE OBRAZU

- Aby uniknąć migotania obrazu podczas jego przerysowywania gdy występuje konieczność jego akutalizacji, stosuje się technikę **podwójnego buforowania** (ang. *double buffering*)
- Mamy wówczas dwa bufory klatki:
  - **Bufor "przedni"** (ang. *front buffer*)
    - Jest wyświetlany
  - **Bufor "tylny"** (ang. *back buffer*)
    - Jest aktualizowany
- Gdy zakończy się aktualizacja back buffera, **zamieniane są one miejscami** (ang. *buffer swap*)



WYŚWIETLANIE OBRAZU

- Dla zachowania wrażenia **płynności** wyświetlanego ruchomego obrazu, należy upewnić się że kolejne klatki animacji będą docierać do monitora, gdy ten **będzie gotowy** je wyświetlić
- W tym celu stosuje się tzw. **synchronizację pionową** (ang. *vertical synchronisation / VSync*)
  - Sterownik karty graficznej **czeka** z zamianą buforów, aż monitor będzie gotowy wyświetlić kolejną klatkę
- **W innym wypadku** pomimo sprzętowej możliwości wygenerowania nawet większej liczby klatek na sekundę niż wyświetla monitor, obraz **nie będzie płynny**

WYŚWIETLANIE OBRAZU

Artefakt tego rodzaju nazywa się często ang. słowem ***tearing***.

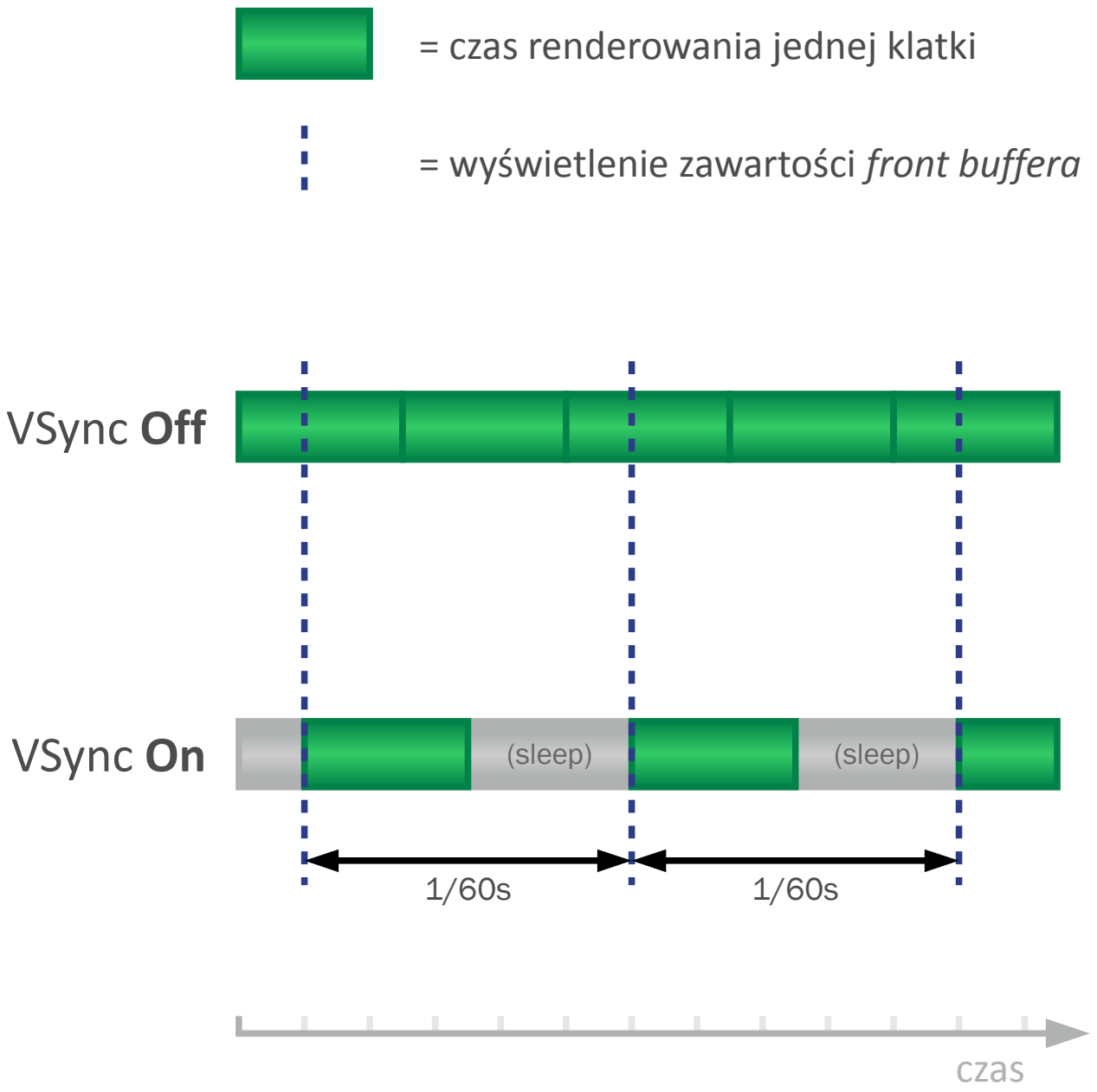
Synchronizacja pionowa **wyłączona**

Synchronizacja pionowa **włączona**

WYŚWIETLANIE OBRAZU

Częstotliwość, z jaką powstają kolejne klatki animacji, a dokładniej **z jaką zamieniane są bufory**, określa się mianem ***frame rate***.

Częstotliwość, z jaką wyświetlacz powraca do ponownego **odświeżania tej samej linii obrazu**, określa się mianem ***refresh rate***.



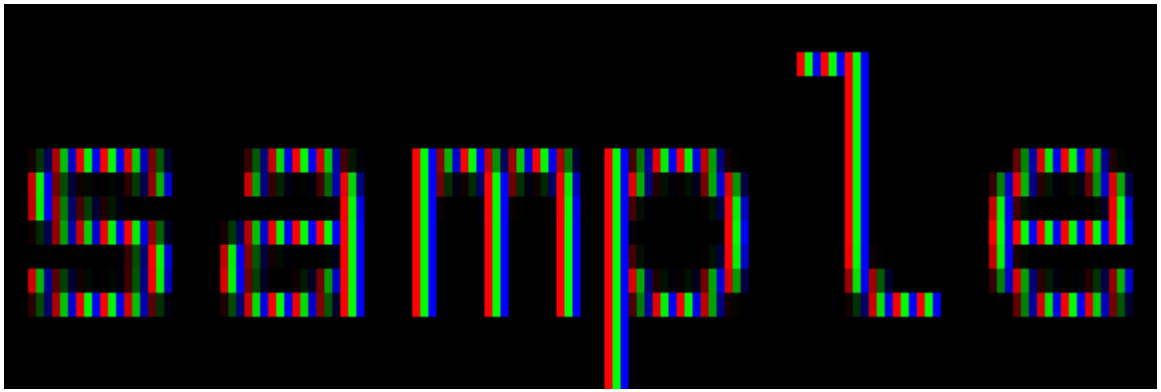
WYŚWIETLANIE OBRAZU

- Technologia **NVIDIA G-Sync**
  - **Synchronizacja wyświetlacza** z kartą graficzną – czyli podejście odwrotne
  - Wyświetlacz ma **adaptacyjną częstotliwość odświeżania**, dostosowującą się do aktualnej liczby klatek na sekundę
    - Zakres 30-144 Hz, zależnie od sprzętu
  - Podejście wbrew pozorom **nie jest nowe**, podobne rozwiązanie *Adaptive Sync* jest częścią standardu DisplayPort 1.2a – zaimplementowane choćby przez AMD w ich *FreeSync*; różnice:
    - G-Sync duplikuje klatki gdy ich częstotliwość poniżej minimum
    - G-Sync zapobiega nadpisywaniu klatek

Źródło obrazu:
NVIDIA

WYŚWIETLANIE OBRAZU

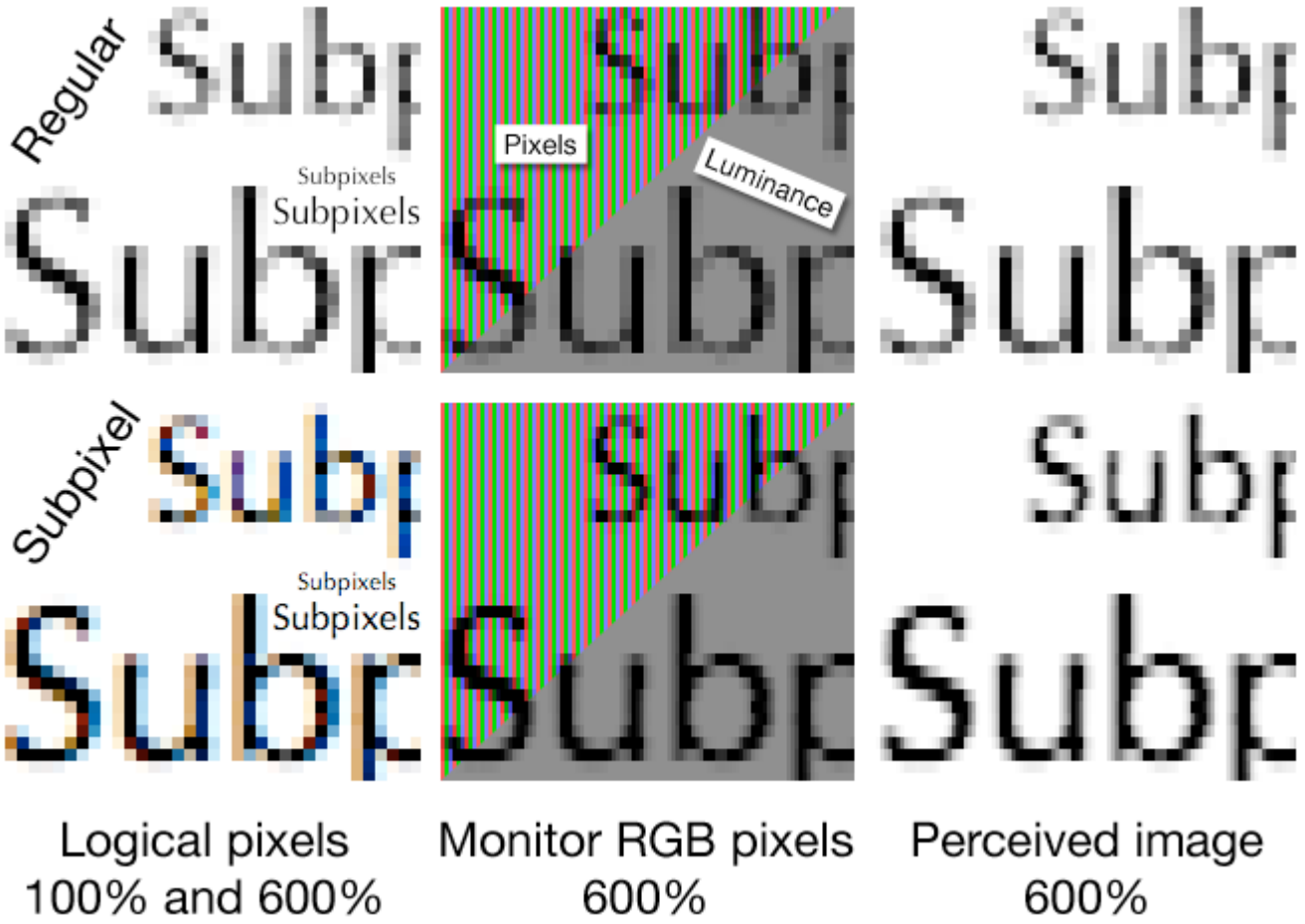
- Subpiksele mogą być wykorzystane dla **poprawienia jakości wyświetlanego tekstu**
  - Np. technologia *Microsoft ClearType*, *Quartz* dla *MacOS*, *FreeType*
  - Sztuczne zwiększenie rozdzielczości obrazu poprzez takie użycie kolorowych subpikseli, by osiągnąć **postrzegalne poziomy luminancji** odpowiadające **poziomom szarości**



Źródło obrazu:
http://www.wikipedia.org

WYŚWIETLANIE OBRAZU

- Przykład działania technologii *Quartz*



Źródło obrazu:
http://www.wikipedia.org

Zachodniopomorski
Uniwersytet
Technologiczny
w Szczecinie

http://bazyluk.net/dydaktyka

Bartosz Bazyluk

Kamera

Kamera w scenie trójwymiarowej. Implementacja kamery FPP.

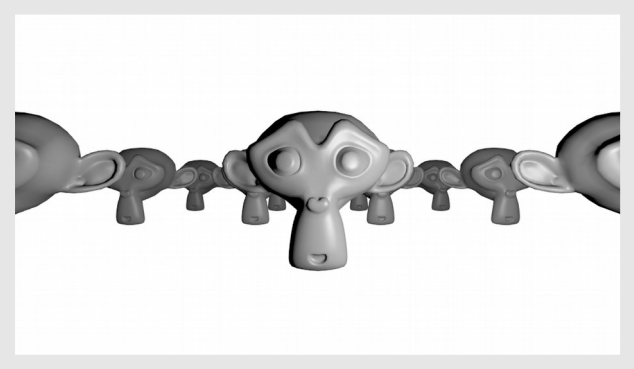
Wydział
Informatyki

Grafika Komputerowa i Wizualizacja, Informatyka S1, II Rok

Kamera

- Jest to wirtualny *koncept* opisujący **sposób oglądania sceny przez obserwatora**, dla którego renderujemy obraz
- Wyróżnia się dwa podstawowe **rodzaje rzutowań**:

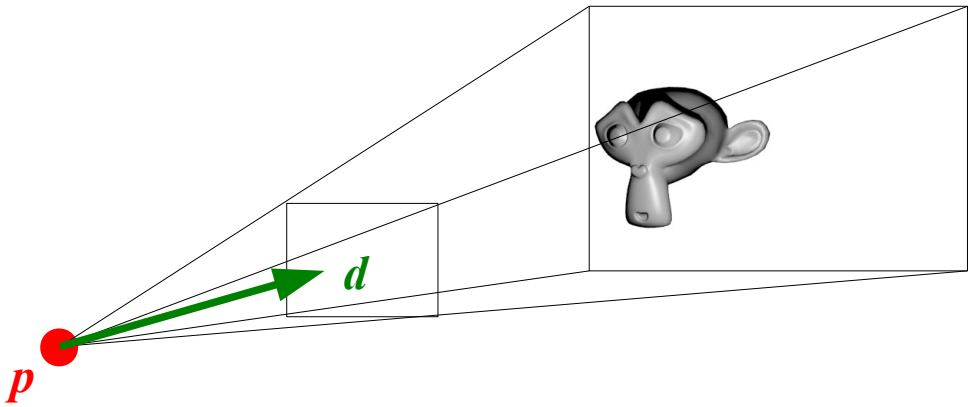
Rzut perspektywiczny



Rzut prostokątny
(ang. *orthographic*)

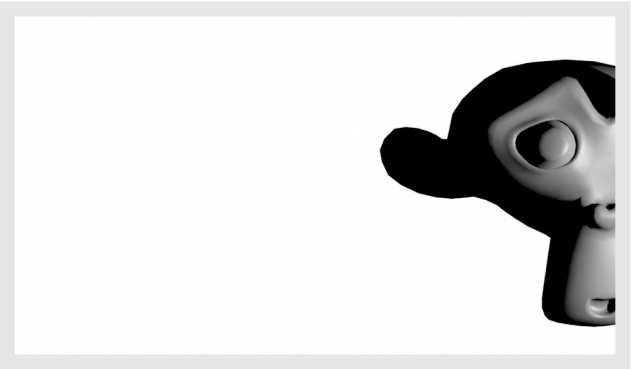
Kamera perspektywiczna

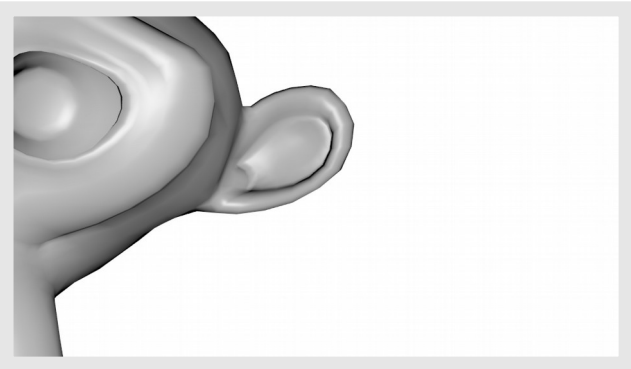
- Kamera musi być **opisana w sposób matematyczny**, aby możliwe było uwzględnienie jej cech w procesie renderowania
- Przede wszystkim musi posiadać zdefiniowane:
  - **Pozycję** (punkt zaczepienia): p
  - **Kierunek widzenia**: d



Kamera perspektywiczna

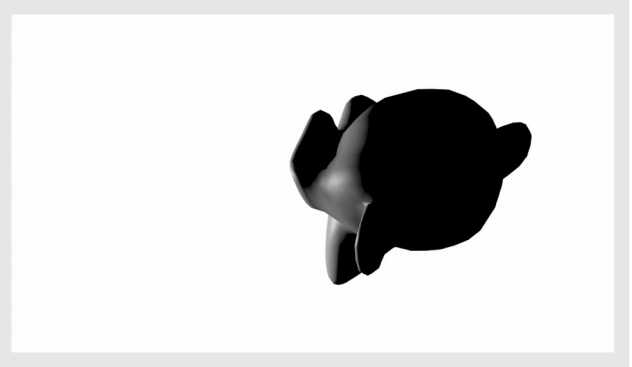
- Położenie w przestrzeni 3D jest po prostu **3-elementowym wektorem**
- Zmiana położenia powoduje **przemieszczenie** naszego wirtualnego "aparatu fotograficznego"





Kamera perspektywiczna

- Kierunek w przestrzeni 3D również jest **3-elementowym wektorem**
  - Kierunek można rozumieć jako wektor leżący na **półprostej** łączącej **punkt zaczepienia** z miejscem, na które kamera ma być **skierowana**
- Kierunek zwyczajowo przechowuje się jako **wektor jednostkowy** ($||d|| = 1$)

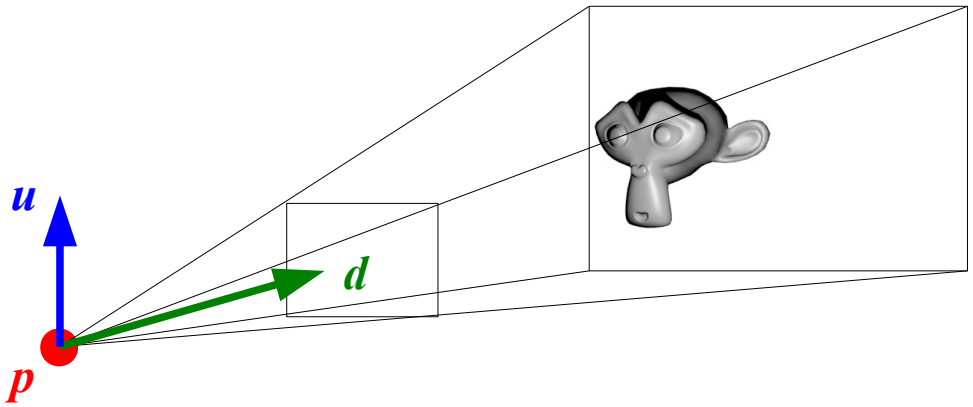


Kamera perspektywiczna

- To jednak **nie wystarczy**, aby **jednoznacznie** określić sposób obserwacji sceny
- Oba poniższe przykłady zostały wyrenderowane **z tego samego miejsca**, **w tym samym kierunku**:

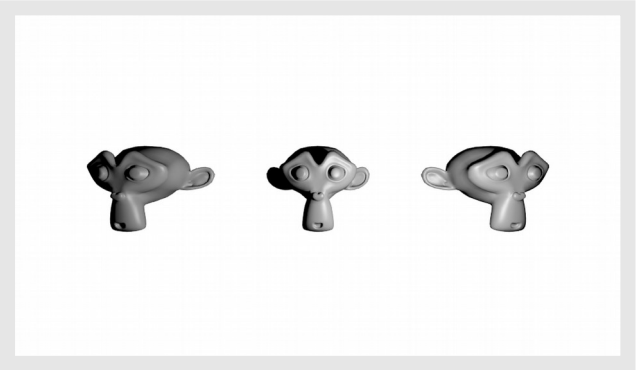
Kamera perspektywiczna

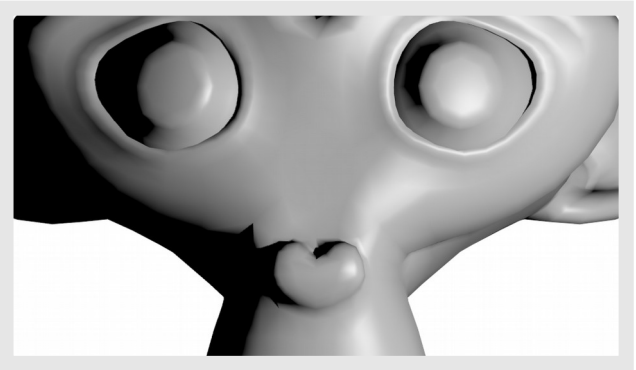
- Konieczne jest wprowadzenie dodatkowo **wektora pionu** (ang. *up vector*): ***u***
- Najczęściej jest to ***(0, 1, 0)*** jeśli za drugą współrzędną przyjmiemy tę skierowaną ku górze świata

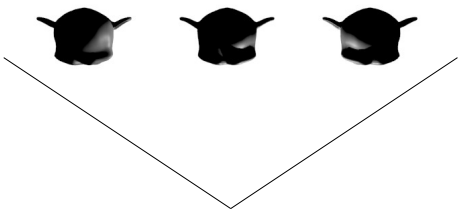


Kamera perspektywiczna

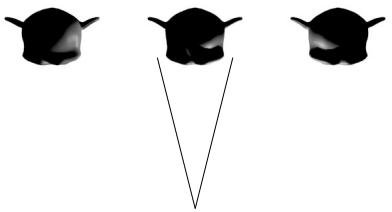
- Dodatkowo kamerę określają:
  - **Kąt widzenia** (najczęściej w stopniach) *fov*
    - Jego zmiana odpowiada **zmianie ogniskowej obiektywu** (popularnie zwanej "*zoomem*" w aparacie fotograficznym)
    - Nie jest tożsamy z przybliżeniem/oddaleniem kamery!







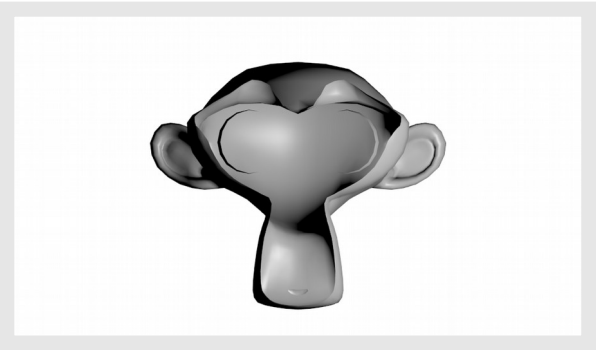
Szeroki kąt
(krótka ogniskowa)

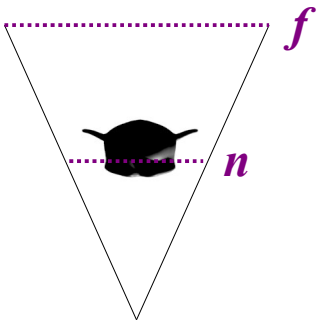


Wąski kąt
(długa ogniskowa)

Kamera perspektywiczna

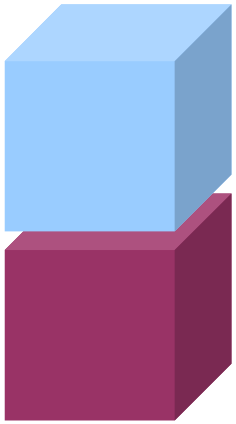
- Dodatkowo kamerę określają:
  - **Odległość płaszczyzn przycinania: bliskiej n i dalekiej f** (ang. *near/far clipping plane*)
    - Określają, od jakiej do jakiej odległości będzie renderowana scena
    - Ma to związek z **precyzją bufora głębokości**
      - Im większy zakres odległości jest renderowany, tym większa szansa powstania problemów typu *z-fighting* – wartości powinny zostać dobrane odpowiednio do charakterystyki danej sceny



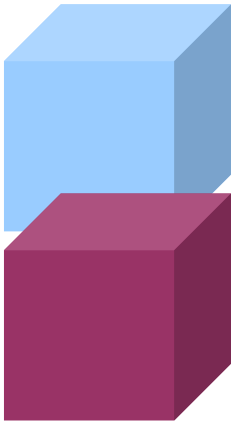


Test głębokości (ang. *depth test*)

- **Algorytm malarza** (ang. *painter's algorithm*)
  - Obiekty przykrywają się **zgodnie z kolejnością rysowania**
  - To, co zostanie narysowane **później**, zawsze przykryje to, co było narysowane **wcześniej**

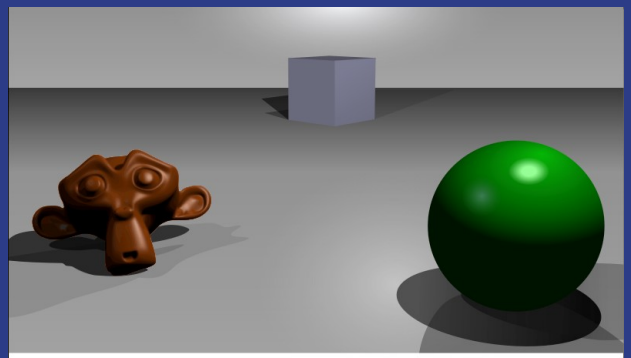


Najpierw narysowano **fioletowy** sześcian

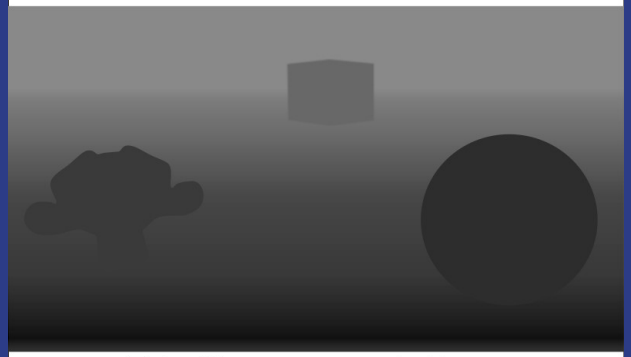


Najpierw narysowano **niebieski** sześcian

Test głębokości
(ang. *depth test*)



A simple three-dimensional scene



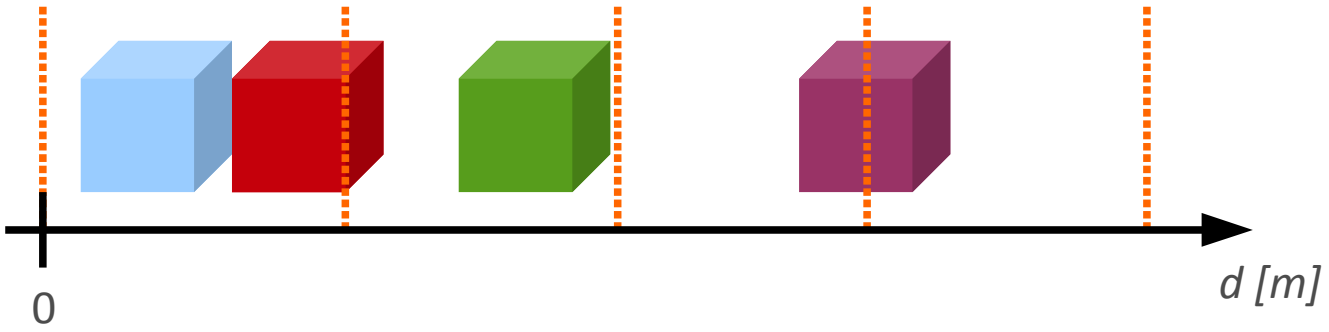
Z-buffer representation

Źródło obrazu:
Wikipedia

- **Algorytm *bufora Z***
  - Oprócz *bufora koloru*, tworzony jest też **bufor głębokości** (*bufor Z*) o odpowiadającym mu rozmiarze
  - W *buforze Z* przechowywane są informacje o **odległości od kamery** tego fragmentu powierzchni obiektu, któremu odpowiada kolor znajdujący się w tym pikselu w buforze koloru
  - Jeśli w wyniku renderowania kolejnego obiektu sceny okaże się, że w danym pikselu bufora już coś się wcześniej znalazło, **wykonywany jest test głębokości**:
    - **Jeśli odległość** od kamery nowego fragmentu powierzchni obiektu **jest mniejsza niż wartość w buforze głębokości**, to nadpisujemy ją nową wartością i nadpisujemy kolor
      - Bo **nowy obiekt jest bliżej kamery**, więc przysłania to, co wcześniej było w buforze koloru
    - **Jeśli odległość** od kamery nowego fragmentu **jest większa niż wartość w buforze głębokości**, to pomijamy nową wartość
      - Bo wcześniej już narysowaliśmy coś, co **przysłania** nasz nowy obiekt

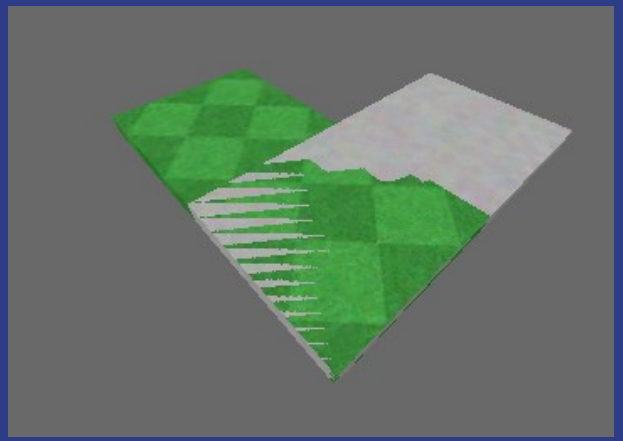
Test głębokości
(ang. *depth test*)

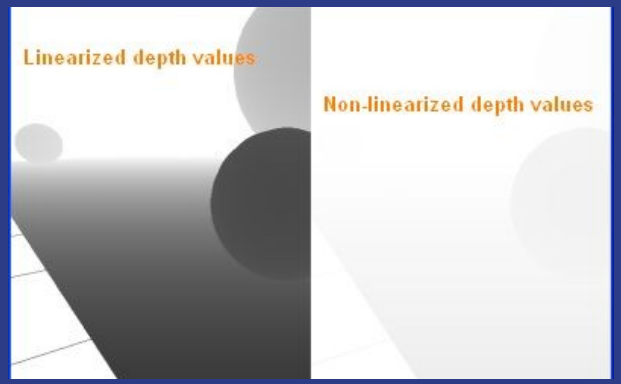
- **Bufor Z**
  - Jest **jednokanałową mapą bitową**, w której każdy na każdy piksel przeznaczono **określoną liczbę bitów**
    - Najczęściej **24 bity** lub **32 bity**
  - W związku z tym możliwe jest zapisanie jedynie **skończonej liczby różnych wartości** odległości od kamery
    - np. wartości całkowite od 0 do 2^{24}-1
  - Prowadzi to do potencjalnych **konfliktów**



Źródło obrazu:
Wikipedia

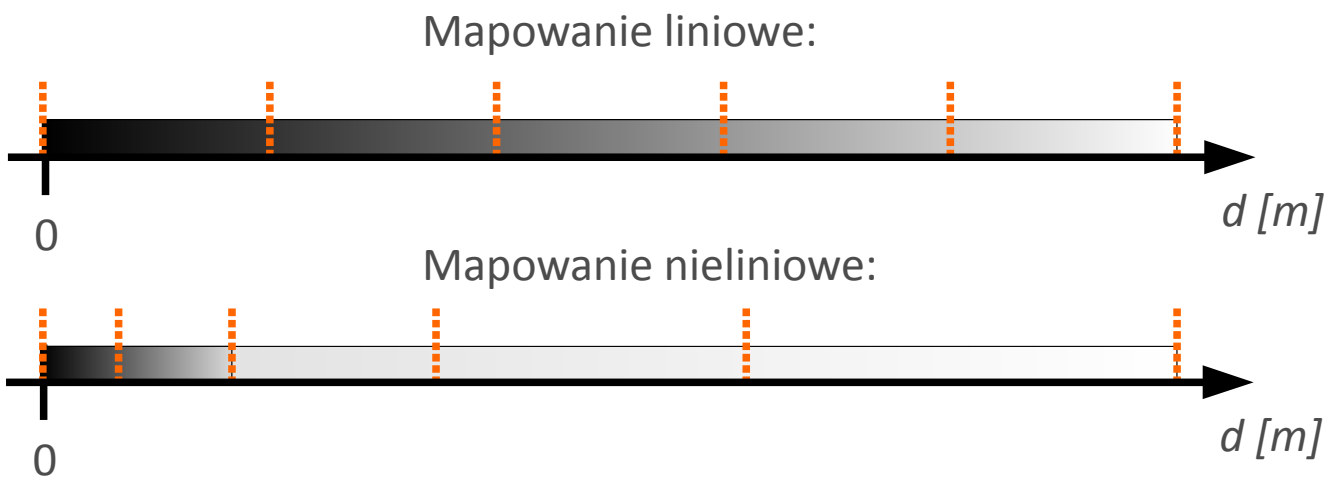
Test głębokości
(ang. *depth test*)





Źródło obrazu:
GeeXLab, JEGX

- **Precyzja** *bufora Z*
  - **Zbyt mała precyzja** prowadzi do **konfliktów**, które objawiają się artefaktem *z-fighting*
  - Należy **ograniczyć zakres odległości**
    - Poprzez użycie **płaszczyzn odcinania**
  - Stosuje się **nieliniowe mapowanie odległości** na wartości *bufora Z*
    - Najczęściej **logarytmiczne**
    - **Większa precyzja dla małych odległości**, mniejsza dla dużych



Kamera perspektywiczna

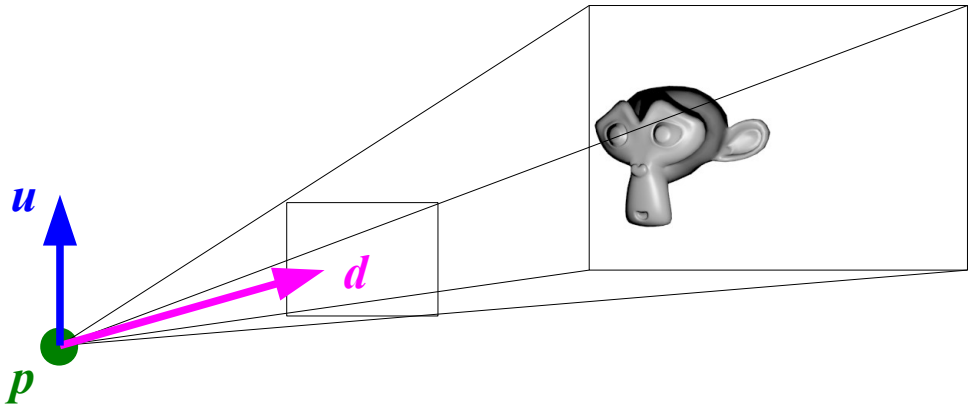
- Dodatkowo kamerę określają:
  - **Proporcje** boków podstawy ściętego ostrosłupa (ang. *frustum*)
    - Stosunek szerokości do wysokości
      - 4:3, 16:9, 16:10, …

Kamera perspektywiczna

- Podsumowując, do **jednoznacznego opisania kamery** potrzebne są wartości następujących atrybutów:
  - Cechy **widoku**:
    - Położenie
    - Kierunek
    - Wektor pionu
  - Cechy **projekcji**:
    - Kąt widzenia
    - Odległości płaszczyzn przycinania
    - Proporcje

Kamera w OpenGL

- Aby opisać jednoznacznie położenie kamery, należy określić:
  - Pozycję ***p***
  - Kierunek widzenia ***d***
  - Wektor pionu ***u***
- Każdy z tych elementów w przestrzeni 3D jest 3-elementowym wektorem

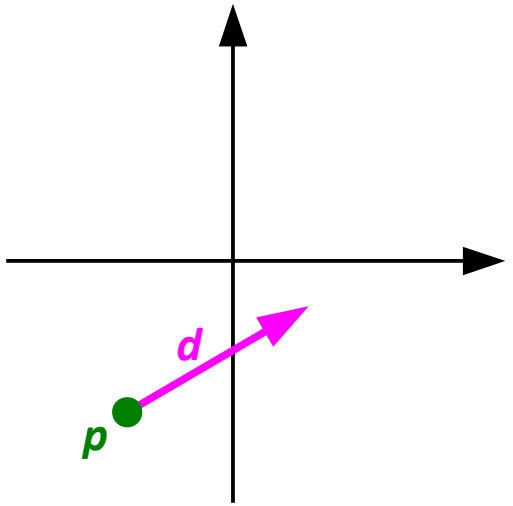


Kamera w OpenGL

- Aby łatwo uzyskać w OpenGL **macierz transformacji** odpowiadającą kamerze **umieszczonej w zadanym punkcie**, patrzącą **w zadanym kierunku** i o **zadanym wektorze pionu**, najlepiej posłużyć się funkcją:
  - **gluLookAt**(
    eyeX, eyeY, eyeZ,
    lookatX, lookatY, lookatZ,
    upX, upY, upZ
)
    - ***eye**** - współrzędne *XYZ* punktu zaczepienia kamery
    - ***lookat**** - współrzędne *XYZ* punktu, na który ma spoglądać kamera
    - ***up**** - współrzędne *XYZ* wektora pionu
      - W dalszych rozważaniach dla uproszczenia przyjmiemy, że wektor pionu zawsze będzie równy *(0;1;0)*

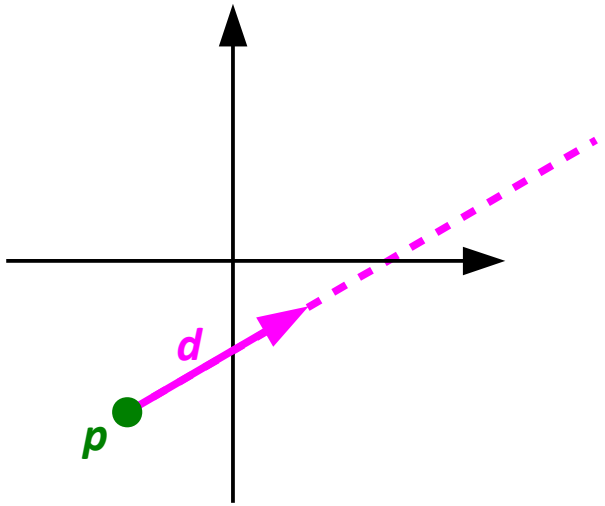
Kamera w OpenGL

- Należy zwrócić uwagę, że *gluLookAt()* nie przyjmuje kierunku widzenia kamery, a punkt na który kamera ma spoglądać
  - Jest to szczególnie istotne, gdy naszym zadaniem jest zaprogramowanie **interaktywnej kamery pierwszoosobowej**
    - W takim przypadku dużo wygodniej jest przechowywać kierunek widzenia jako **wektor jednostkowy**
    - Na podstawie **pozycji** kamery o raz **kierunku** można **łatwo uzyskać współrzędne punktu**, który można przekazać do *gluLookAt()*.



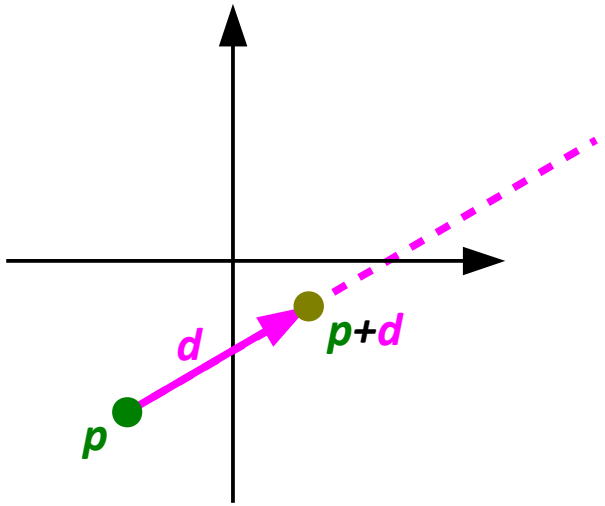
Kamera w OpenGL

- Należy zwrócić uwagę, że *gluLookAt()* nie przyjmuje kierunku widzenia kamery, a punkt na który kamera ma spoglądać
  - Jest to szczególnie istotne, gdy naszym zadaniem jest zaprogramowanie **interaktywnej kamery pierwszoosobowej**
    - W takim przypadku dużo wygodniej jest przechowywać kierunek widzenia jako **wektor jednostkowy**
    - Na podstawie **pozycji** kamery o raz **kierunku** można **łatwo uzyskać współrzędne punktu**, który można przekazać do *gluLookAt()*.



Kamera w OpenGL

- Należy zwrócić uwagę, że *gluLookAt()* nie przyjmuje kierunku widzenia kamery, a punkt na który kamera ma spoglądać
  - Jest to szczególnie istotne, gdy naszym zadaniem jest zaprogramowanie **interaktywnej kamery pierwszoosobowej**
    - W takim przypadku dużo wygodniej jest przechowywać kierunek widzenia jako **wektor jednostkowy**
    - Na podstawie **pozycji** kamery o raz **kierunku** można **łatwo uzyskać współrzędne punktu**, który można przekazać do *gluLookAt()*.



Stan kamery

- Reprezentacja stanu kamery w pamięci
  - Konieczne jest zapamiętanie **trzech wektorów 3-elementowych**:
    - Położenia
    - Kierunku
    - Pionu
  - Sugerowane jest utworzenie **struktury** przechowującej trzy składowe *XYZ*:

```
struct vec3 {
        float x, y, z;
};
```

  - Wówczas **stan kamery** można przedstawić następująco:

```
struct SCameraState {
        vec3 pos;
        vec3 dir;
        vec3 up;
};
```

Stan kamery

- Jako że dla uproszczenia przyjęliśmy, że **wektor pionu jest stały**, możemy go pominąć
- Przydatne może okazać się przechowanie **aktualnej prędkości** przesuwania kamery
  - Np. płynne wygaszanie ruchu
- Zatem struktura stanu kamery może przyjąć następującą postać:

```
struct SCameraState {
        vec3 pos;
        vec3 dir;
        float speed;
};
```

- Na potrzeby zadania utworzymy globalną instancję tej struktury i nazwiemy ją *player*

```
SCameraState player;
```

Stan kamery

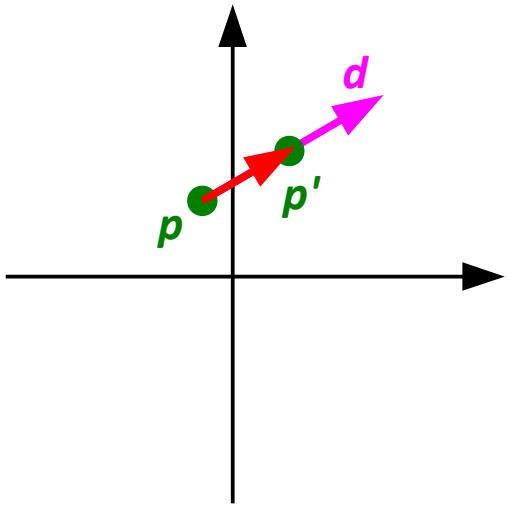
- W takiej sytuacji, możemy każdorazowo zasilać wywołanie *gluLookAt()* następującymi danymi:

```
gluLookAt(
        player.pos.x,
        player.pos.y,
        player.pos.z,
        player.pos.x + player.dir.x,
        player.pos.y + player.dir.y,
        player.pos.z + player.dir.z,
        0,
        1,
        0
);
```

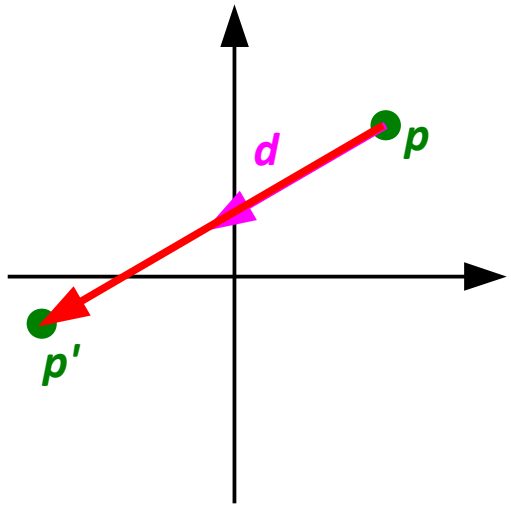
Ruch kamery w przód/tył

- Aby zrealizować taki ruch, trzeba zmodyfikować położenie punktu zaczepienia kamery
- "Przód" w danym momencie odpowiada wektorowi kierunku kamery
- Pozycję po przesunięciu obliczamy w prosty sposób:

$$p' = p + speed \cdot d$$



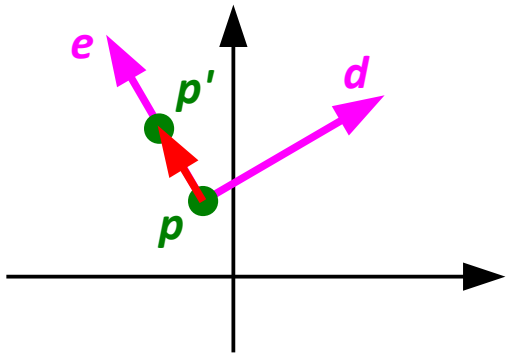
Mała prędkość (mnożnik)



Duża prędkość (mnożnik)

Ruch kamery w bok

- Aby uwzględnić kierunek widzenia kamery, ruszając się na boki należy poruszać się po **kierunku prostopadłym**
  - W założeniu mamy poruszać się tylko **po płaszczyźnie *XZ***



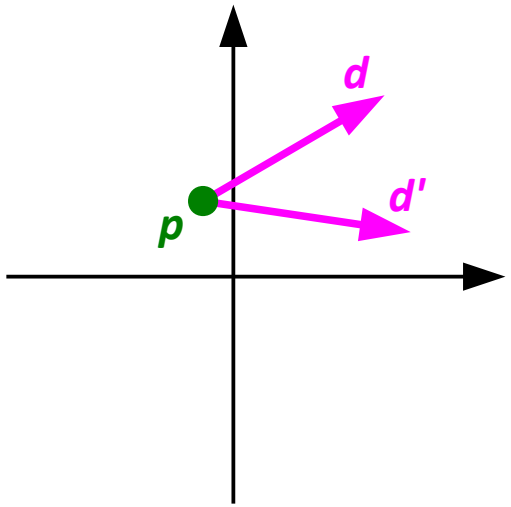
$$p' = p + speed \cdot e, \quad e \perp d$$

- Jak wyznaczyć **wektor prostopadły do danego**?

$$d = \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix} \Rightarrow e = \begin{pmatrix} -d_z \\ 0 \\ d_x \end{pmatrix}$$

Obrót kamery

- Obrót kamery jest zmianą kierunku jej widzenia, bez wpływu na punkt zaczepienia
  - W założeniu mamy poruszać się tylko **po płaszczyźnie *XZ***
  - Rezultatem jest wektor kierunku **obrócony o zadany kąt**
    - **Jak wyliczyć jego współrzędne?**

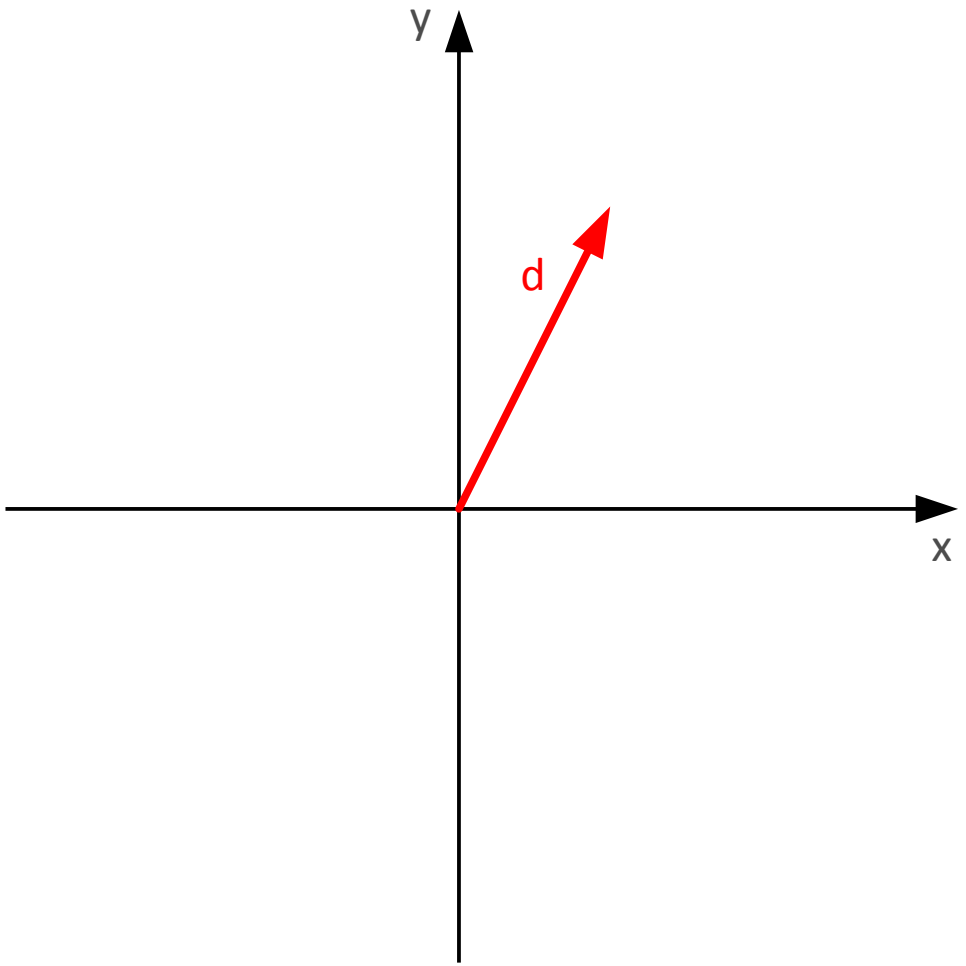


Obrót wektora kierunku w 2D

Dla uproszczenia sprowadźmy problem do **dwóch wymiarów**.

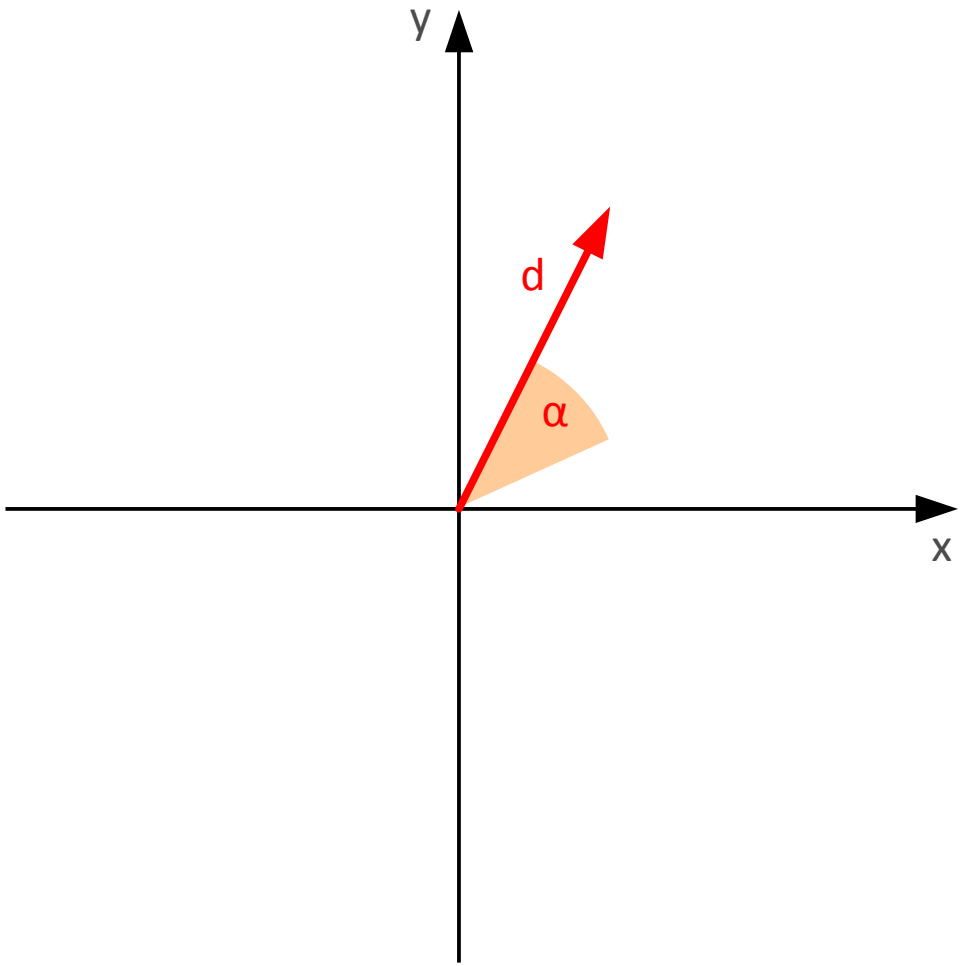
Będziemy dokonywać obrotu tylko **w obrębie jednej płaszczyzny**, którą roboczo nazwiemy *x/y*.

Chcąc poruszać się po płaszczyźnie *x/z* układu współrzędnych *OpenGL* należy odpowiednio zmienić oznaczenie osi!



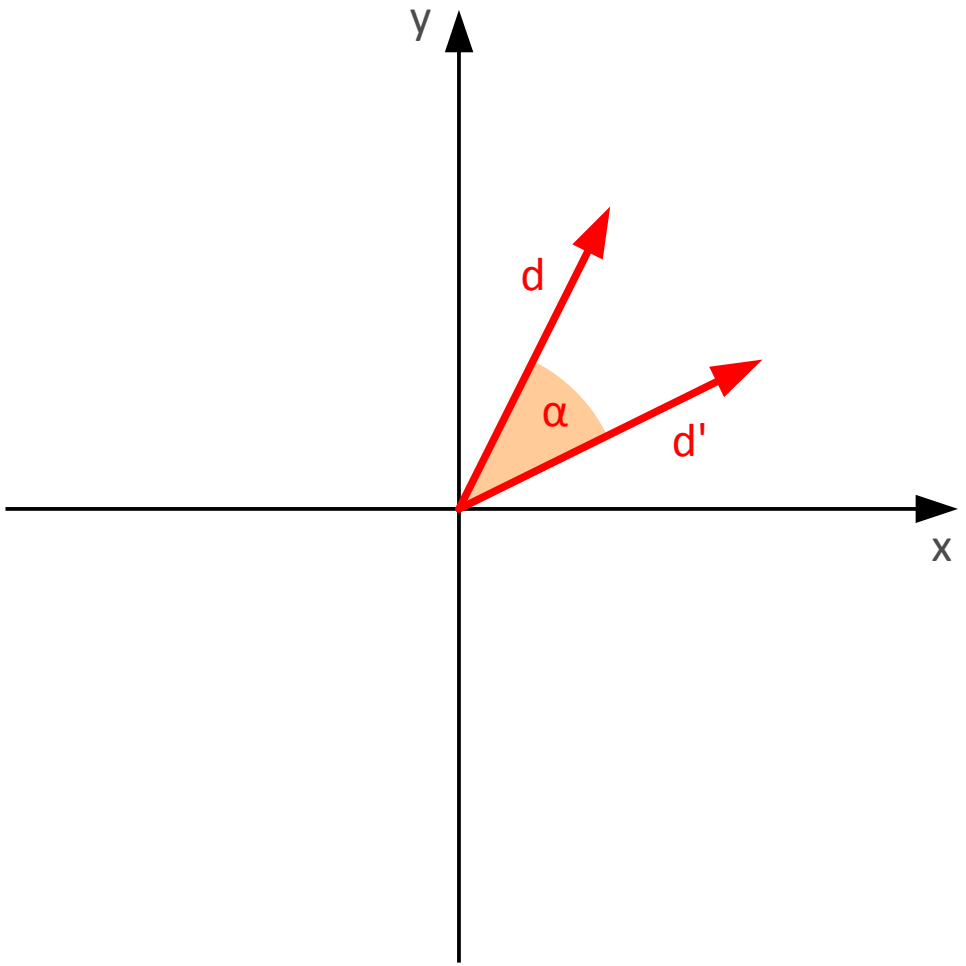
Obrót wektora kierunku w 2D

Chcemy dokonać **obrotu** kamery o **zadany kąt α**.



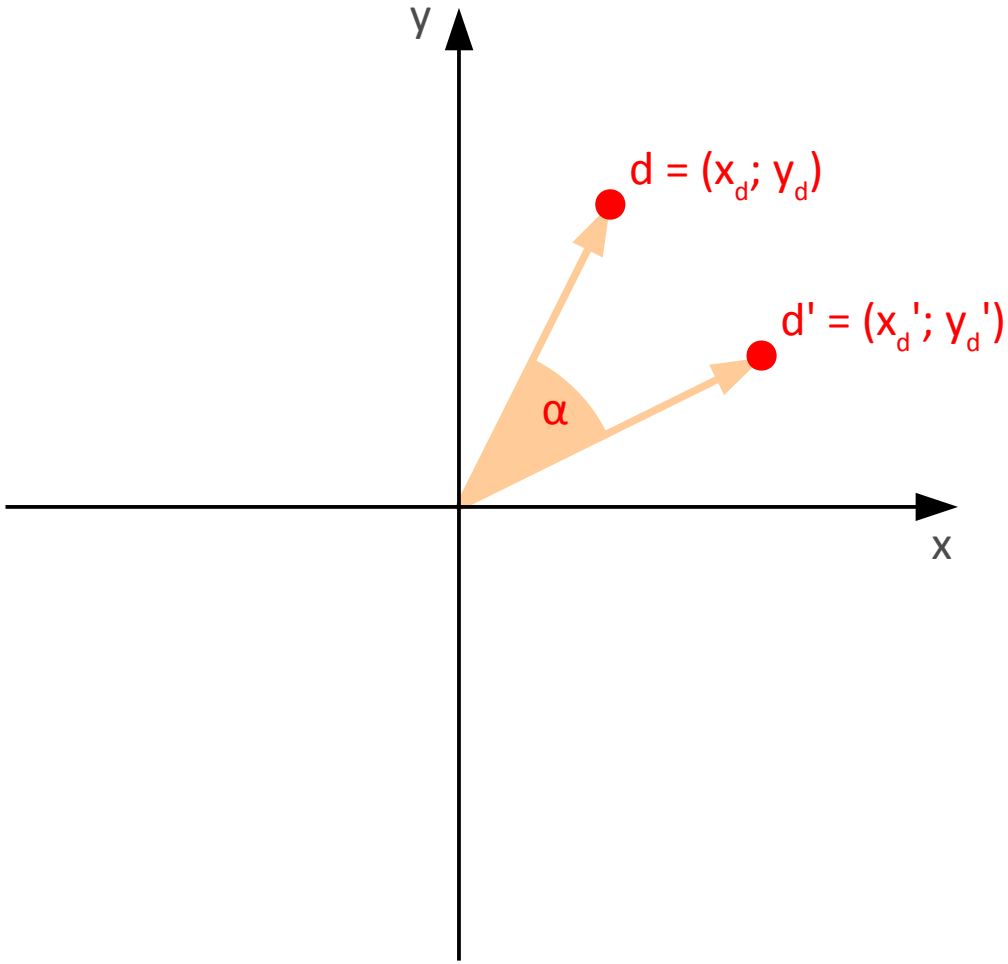
Obrót wektora kierunku w 2D

Naszym wynikiem powinien być **nowy wektor kierunku *d'***, uzyskany po obróceniu pierwotnego wektora *d* wokół początku układu współrzędnych.



Obrót wektora kierunku w 2D

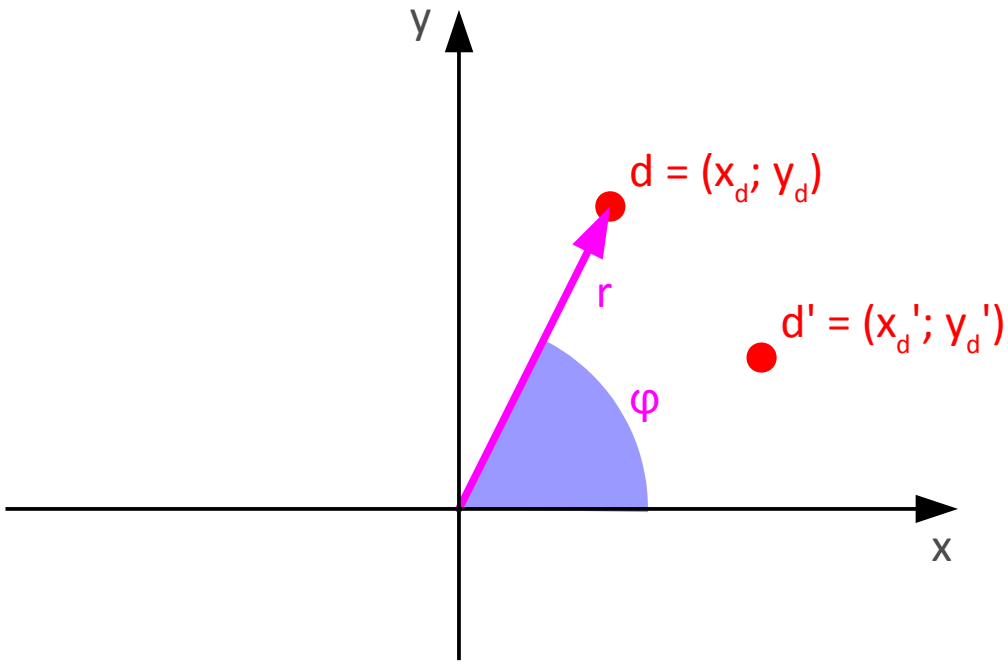
Można przyjąć, że wektory zaczepione w początku układu współrzędnych to **położenia punktów** znajdujących się na ich końcach.



Obrót wektora kierunku w 2D

Inny sposób opisania położenia punktu na płaszczyźnie: **współrzędne biegunowe** (ang. *polar coordinates*).

Położenie definiuje się jako **odległość od bieguna** *(u nas: pocz. układu współrzędnych)* oraz **kąt pomiędzy osią biegunową** *(u nas: +y)*, **a promieniem wodzącym**.



$$d = (x_d ; y_d) = (r ; \varphi)$$

współrzędne kartezjańskie

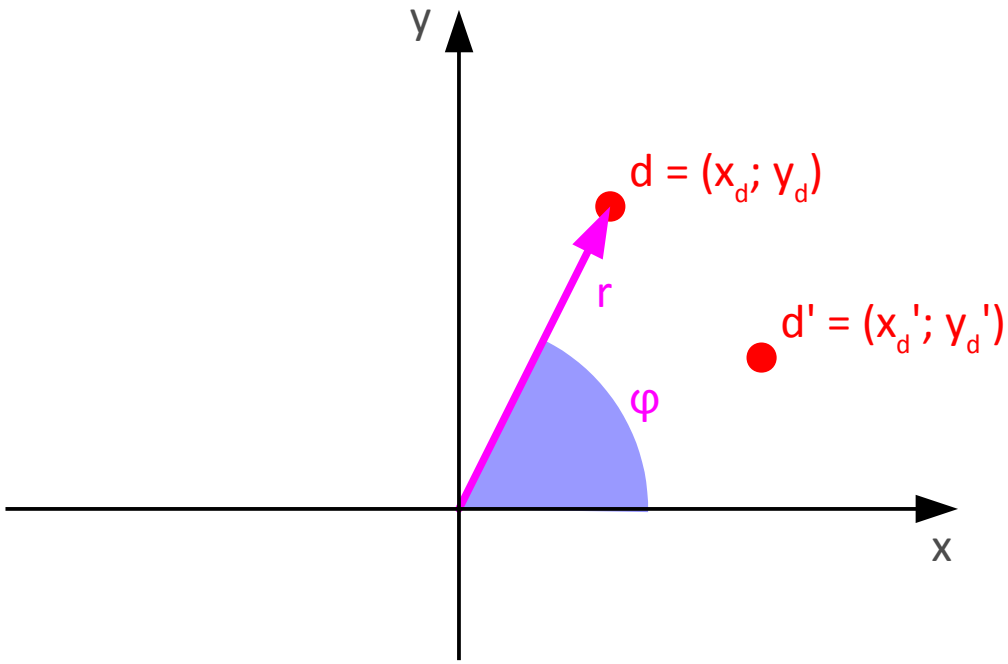
współrzędne biegunowe

Obrót wektora kierunku w 2D

Aby **wyliczyć współrzędne biegunowe odpowiadające kartezjańskim**, wystarczy posłużyć się dwoma prostymi wzorami.

Funkcja cyklometryczna *atan2* jest dwuargumentową odmianą funkcji *arcus tangens*, biorącą pod uwagę ćwiartkę układu w której znajduje się zadany kąt.

[http://en.wikipedia.org/wiki/Atan2]

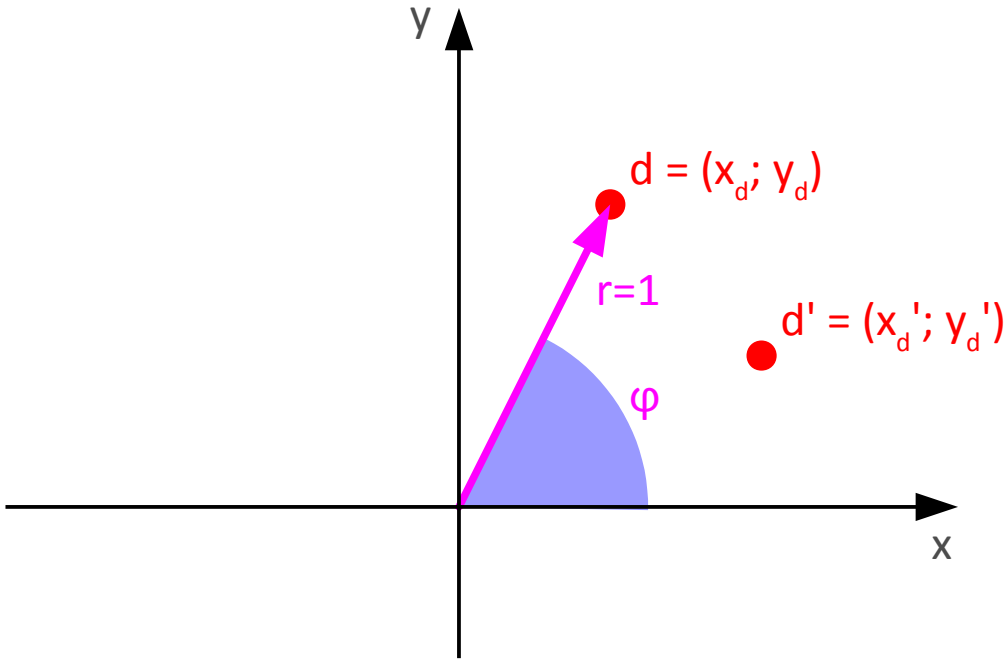


$$d = (x_d ; y_d) = (r ; \varphi)$$

$$r = \sqrt{x^2 + y^2}$$
$$\varphi = atan2(y , x)$$

Obrót wektora kierunku w 2D

Pamiętamy, że nasz wektor kierunku jest zawsze **wektorem jednostkowym**. W takim razie długość promienia wodzącego *r* będzie **zawsze równa 1**.



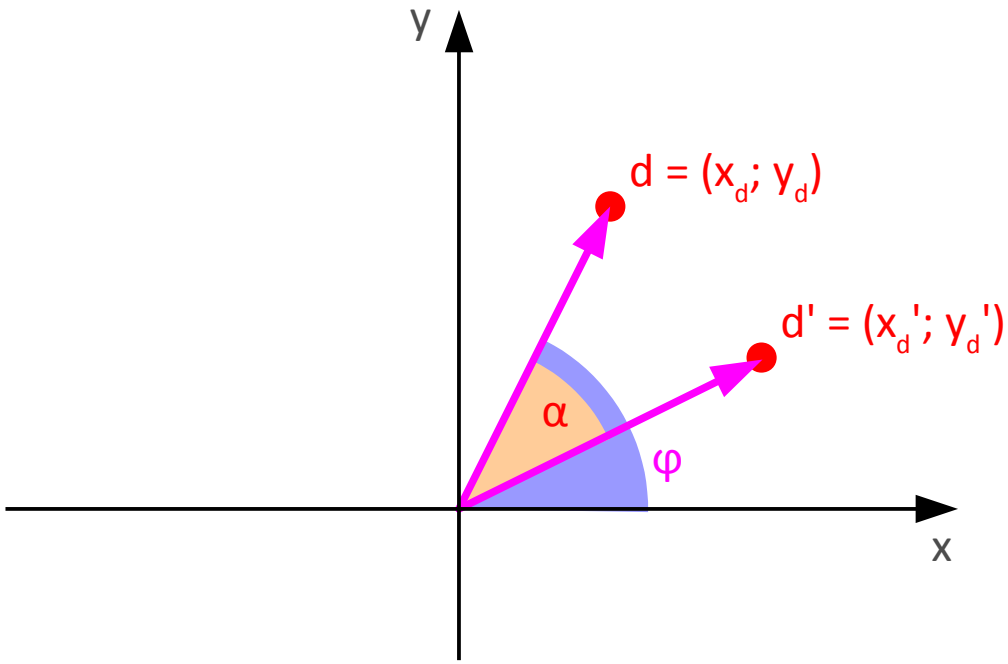
$$d=(x_d\,;y_d)=(r\,;\varphi)=(1\,;\varphi)$$

$$r=\sqrt{x^2+y^2}=\mathbf{1}$$
$$\varphi=atan2(y\,,x)$$

Obrót wektora kierunku w 2D

Zwróćmy uwagę, że we współrzędnych biegunowych podczas obracania wektora, **ulegnie zmianie tylko jedna współrzędna**.

Nowy kąt zostanie powiększony/pomniejszony o wartość kąta α, o który dokonujemy obrotu.



$$d=(x_d\,;y_d)=(r\,;\varphi)=(1\,;\varphi)$$

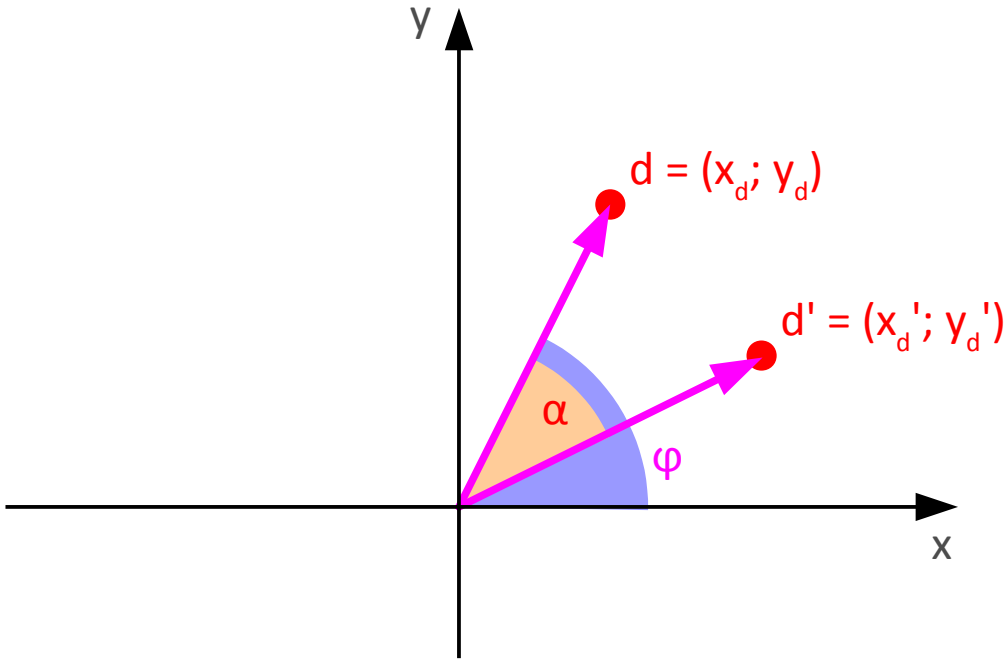
$$r=\sqrt{x^2+y^2}=1$$
$$\varphi=atan\,2\,(y\,,x)$$

$$d'=(1\,;\varphi\pm\alpha)$$

Obrót wektora kierunku w 2D

Mając współrzędne biegunowe wektora po dokonaniu obrotu, wystarczy **powrócić do kartezjańskiego układu współrzędnych**.

Służą do tego dwa proste wzory.



$$d=(x_d\,;y_d)=(r\,;\varphi)=(1\,;\varphi)$$

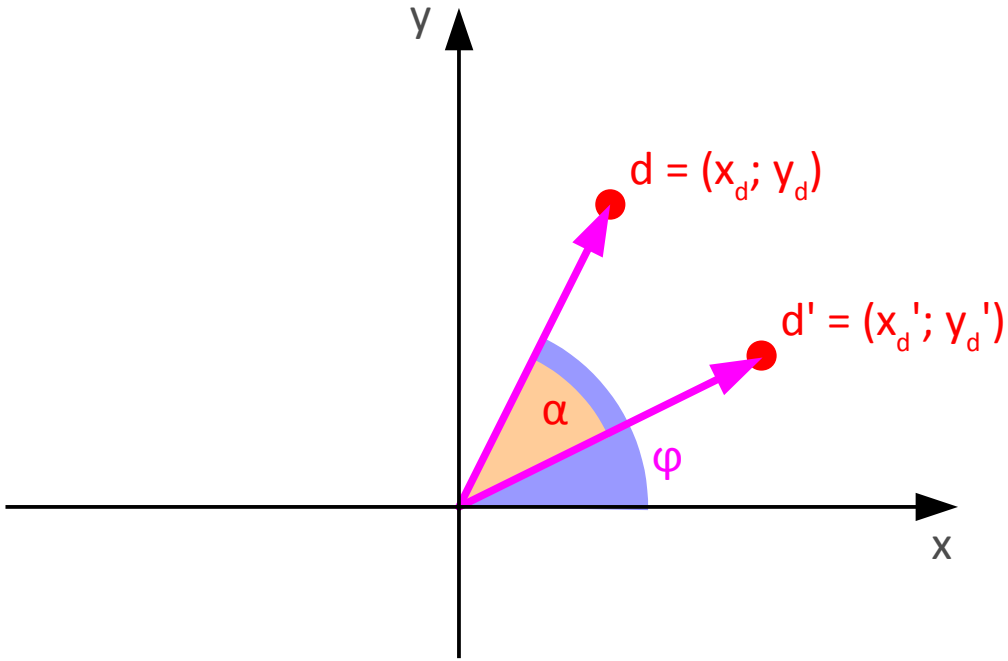
$$r=\sqrt{x^2+y^2}=1$$
$$\varphi=atan\,2\,(y\,,x)$$

$$d\,'=(1\,;\varphi\pm\alpha)$$

$$x=r\cos(\varphi\pm\alpha)$$
$$y=r\sin(\varphi\pm\alpha)$$

Obrót wektora kierunku w 2D

Należy pamiętać, że *r=1*, przez co sprawa jest jeszcze prostsza.



$$d=(x_d\,;y_d)=(r\,;\varphi)=(1\,;\varphi)$$

$$r=\sqrt{x^2+y^2}=\mathbf{1}$$
$$\varphi=atan\,2\,(y\,,x)$$

$$d\,'=(1\,;\varphi\pm\alpha)=\mathbf{(\cos(\varphi\pm\alpha);\sin(\varphi\pm\alpha))}$$

$$x=r\cos(\varphi\pm\alpha)$$
$$y=r\sin(\varphi\pm\alpha)$$

Zachodniopomorski
Uniwersytet
Technologiczny
w Szczecinie

http://bazyluk.net/dydaktyka

Bartosz Bazyluk

Kamera

Kamery w scenie trójwymiarowej. Implementacja kamery FPP.

Wydział
Informatyki

Grafika Komputerowa i Wizualizacja, Informatyka S1, II Rok