# Számítógépes Grafika

Bán Róbert
robert.ban102+cg@gmail.com

Eötvös Loránd Tudományegyetem
Informatikai Kar

2021-2022. tavaszi félév

# Tartalom

# Tartalom

# Emlékeztető

- Múlt órán megismerkedtünk a rekurzív sugárkövetéssel

# Emlékeztető

- Múlt órán megismerkedtünk a rekurzív sugárkövetéssel
- Előnyei:

# Emlékeztető

- Múlt órán megismerkedtünk a rekurzív sugárkövetéssel
- Előnyei:
  - A színtér benépesítésére minden használható, ami metszhető sugárral

# Emlékeztető

- Múlt órán megismerkedtünk a rekurzív sugárkövetéssel
- Előnyei:
  - A színtér benépesítésére minden használható, ami metszhető sugárral
  - Rekurzióval könnyen implementálható programmal

# Emlékeztető

- Múlt órán megismerkedtünk a rekurzív sugárkövetéssel
- Előnyei:
  - A színtér benépesítésére minden használható, ami metszhető sugárral
  - Rekurzióval könnyen implementálható programmal
  - A fényt részecskeként kezeli, az ebből következő hatások könnyen megjeleníthetőek

# Emlékeztető

- Múlt órán megismerkedtünk a rekurzív sugárkövetéssel
- Előnyei:
  - A színtér benépesítésére minden használható, ami metszhető sugárral
  - Rekurzióval könnyen implementálható programmal
  - A fényt részecskeként kezeli, az ebből következő hatások könnyen megjeleníthetőek
- Ugyanakkor láttuk, hogy vannak hátrányai is:

# Emlékeztető

- Múlt órán megismerkedtünk a rekurzív sugárkövetéssel
- Előnyei:
  - A színtér benépesítésére minden használható, ami metszhető sugárral
  - Rekurzióval könnyen implementálható programmal
  - A fényt részecskeként kezeli, az ebből következő hatások könnyen megjeleníthetőek
- Ugyanakkor láttuk, hogy vannak hátrányai is:
  - Minden pixelnél minden primitívvel kellett tesztelnünk

# Emlékeztető

- Múlt órán megismerkedtünk a rekurzív sugárkövetéssel
- Előnyei:
  - A színtér benépesítésére minden használható, ami metszhető sugárral
  - Rekurzióval könnyen implementálható programmal
  - A fényt részecskeként kezeli, az ebből következő hatások könnyen megjeleníthetőek
- Ugyanakkor láttuk, hogy vannak hátrányai is:
  - Minden pixelnél minden primitívvel kellett tesztelnünk $\rightarrow$ ezen próbáltunk gyorsítani (dobozolás, térfelosztás)

# Emlékeztető

- Múlt órán megismerkedtünk a rekurzív sugárkövetéssel
- Előnyei:
  - A színtér benépesítésére minden használható, ami metszhető sugárral
  - Rekurzióval könnyen implementálható programmal
  - A fényt részecskeként kezeli, az ebből következő hatások könnyen megjeleníthetőek
- Ugyanakkor láttuk, hogy vannak hátrányai is:
  - Minden pixelnél minden primitívvel kellett tesztelnünk $\rightarrow$ ezen próbáltunk gyorsítani (dobozolás, térfelosztás)
  - Globális jellegű az algoritmus, nehezen gyorsítható hardveresen

# Emlékeztető

- Múlt órán megismerkedtünk a rekurzív sugárkövetéssel
- Előnyei:
  - A színtér benépesítésére minden használható, ami metszhető sugárral
  - Rekurzióval könnyen implementálható programmal
  - A fényt részecskeként kezeli, az ebből következő hatások könnyen megjeleníthetőek
- Ugyanakkor láttuk, hogy vannak hátrányai is:
  - Minden pixelnél minden primitívvel kellett tesztelnünk $\rightarrow$ ezen próbáltunk gyorsítani (dobozolás, térfelosztás)
  - Globális jellegű az algoritmus, nehezen gyorsítható hardveresen
  - A fény hullám természetéből adódó jelenségeket nem tudja visszaadni

# Emlékeztető

- Múlt órán megismerkedtünk a rekurzív sugárkövetéssel
- Előnyei:
  - A színtér benépesítésére minden használható, ami metszhető sugárral
  - Rekurzióval könnyen implementálható programmal
  - A fényt részecskeként kezeli, az ebből következő hatások könnyen megjeleníthetőek
- Ugyanakkor láttuk, hogy vannak hátrányai is:
  - Minden pixelnél minden primitívvel kellett tesztelnünk $\rightarrow$ ezen próbáltunk gyorsítani (dobozolás, térfelosztás)
  - Globális jellegű az algoritmus, nehezen gyorsítható hardveresen
  - A fény hullám természetéből adódó jelenségeket nem tudja visszaadni
  - Valósidejű alkalmazásokhoz túl lassú

# Raycasting

```
Minden pixelre a képernyőn:
    Minden objektumra a színtérben:
        Eltalálja az objektumot a pixel sugara?
```

# Inkrementális képszintézis

```
Minden objektumra a színtérben:
    Minden pixelre a képernyőn:
        A pixelt lefedi az objektum vetülete?
```

# Valósidejű grafika

- Sugárkövetésnél tehát ez volt: $\forall$ pixelre indítsunk sugarat: $\forall$ objektummal (geometriával) nézzük van-e metszés

# Valósidejű grafika

- Sugárkövetésnél tehát ez volt: $\forall$ pixelre indítsunk sugarat: $\forall$ objektummal (geometriával) nézzük van-e metszés
- Ehelyett próbáljuk meg ezt: $\forall$ objektumra (geometriára): számoljuk ki, mely pixelekre képeződik le és végül csak a legközelebbit jelenítsük meg!

# Valósidejű grafika

- Sugárkövetésnél tehát ez volt: $\forall$ pixelre indítsunk sugarat: $\forall$ objektummal (geometriával) nézzük van-e metszés
- Ehelyett próbáljuk meg ezt: $\forall$ objektumra (geometriára): számoljuk ki, mely pixelekre képeződik le és végül csak a legközelebbit jelenítsük meg!
- A sebesség nagyban függ attól, hogy milyen gyorsan dönthető el egy pixelről, hogy lefedi-e az objektum vagy sem

# Valósidejű grafika

- Sugárkövetésnél tehát ez volt: $\forall$ pixelre indítsunk sugarat: $\forall$ objektummal (geometriával) nézzük van-e metszés
- Ehelyett próbáljuk meg ezt: $\forall$ objektumra (geometriára): számoljuk ki, mely pixelekre képeződik le és végül csak a legközelebbit jelenítsük meg!
- A sebesség nagyban függ attól, hogy milyen gyorsan dönthető el egy pixelről, hogy lefedi-e az objektum vagy sem
- Emiatt az objektumokat csak egyszerű geometriákból építhetjük fel $\Rightarrow$ ez a gyakorlatban lineáris elemeket jelent (szakaszok és háromszögek)

# Valósidejű grafika

- Sugárkövetésnél tehát ez volt: $\forall$ pixelre indítsunk sugarat: $\forall$ objektummal (geometriával) nézzük van-e metszés
- Ehelyett próbáljuk meg ezt: $\forall$ objektumra (geometriára): számoljuk ki, mely pixelekre képeződik le és végül csak a legközelebbit jelenítsük meg!
- A sebesség nagyban függ attól, hogy milyen gyorsan dönthető el egy pixelről, hogy lefedi-e az objektum vagy sem
- Emiatt az objektumokat csak egyszerű geometriákból építhetjük fel $\Rightarrow$ ez a gyakorlatban lineáris elemeket jelent (szakaszok és háromszögek)
- Minden más geometriát (például gömb) ezekkel a *primitív geometriákkal* közelítjük, **tesszelláljuk**

# Valósidejű grafika – ötletek

- Ne számoljunk feleslegesen sem: amint lehet, szűrjük ki azokat a geometriákat, amelyek biztosan nem képeződnek le a képernyőre

# Valósidejű grafika – ötletek

- Ne számoljunk feleslegesen sem: amint lehet, szűrjük ki azokat a geometriákat, amelyek biztosan nem képeződnek le a képernyőre
- Ezen kívül minden műveletet olyan koordináta-rendszerben végezzünk el, amiben a legkönnyebb végigszámolni

# Valósidejű grafika – ötletek

- Ne számoljunk feleslegesen sem: amint lehet, szűrjük ki azokat a geometriákat, amelyek biztosan nem képeződnek le a képernyőre
- Ezen kívül minden műveletet olyan koordináta-rendszerben végezzünk el, amiben a legkönnyebb végigszámolni
- A korábbi számításaink eredményeit pedig használjuk fel, ahol csak tudjuk

# Inkrementális képszintézis – fogalmak

- *Koherencia*: Pixelek helyett nagyobb logikai egységekből, *primitívekből* indulunk ki

# Inkrementális képszintézis – fogalmak

- *Koherencia*: Pixelek helyett nagyobb logikai egységekből, *primitívekből* indulunk ki
- Pontosság: *objektum tér pontosság* („pixel pontosság" helyett)

# Inkrementális képszintézis – fogalmak

- *Koherencia*: Pixelek helyett nagyobb logikai egységekből, *primitívekből* indulunk ki
- Pontosság: *objektum tér pontosság* („pixel pontosság" helyett)
- *Vágás*: képernyőről (látótérből) kilógó elemeket távolítsuk el, ne számoljunk velük fölöslegesen

# Inkrementális képszintézis – fogalmak

- *Koherencia*: Pixelek helyett nagyobb logikai egységekből, *primitívekből* indulunk ki
- Pontosság: *objektum tér pontosság* („pixel pontosság" helyett)
- *Vágás*: képernyőről (látótérből) kilógó elemeket távolítsuk el, ne számoljunk velük fölöslegesen
- *Inkrementális elv*: az árnyalási és takarási feladatnál kihasználjuk a nagyobb egységenként szerzett információkat (például a háromszögek meredekségét $(\partial_x z, \partial_y z)$ a fragmentek mélységének kiszámítására).

# Összehasonlítás

## Sugárkövetés

- pixelenként számol

## Inkrementális képszintézis

- primitívenként számol

# Összehasonlítás

## Sugárkövetés

- pixelenként számol
- amit lehet sugárral metszeni, az használható

## Inkrementális képszintézis

- primitívenként számol
- ami nem primitív, azt azzal kell közelíteni

# Összehasonlítás

## Sugárkövetés

- pixelenként számol
- amit lehet sugárral metszeni, az használható
- van tükröződés, fénytörés, vetett árnyékok

## Inkrementális képszintézis

- primitívenként számol
- ami nem primitív, azt azzal kell közelíteni
- külön algoritmus kell ezekhez

# Összehasonlítás

## Sugárkövetés

- pixelenként számol
- amit lehet sugárral metszeni, az használható
- van tükröződés, fénytörés, vetett árnyékok
- takarási feladat triviális

## Inkrementális képszintézis

- primitívenként számol
- ami nem primitív, azt azzal kell közelíteni
- külön algoritmus kell ezekhez
- külön meg kell oldani

# Összehasonlítás

## Sugárkövetés

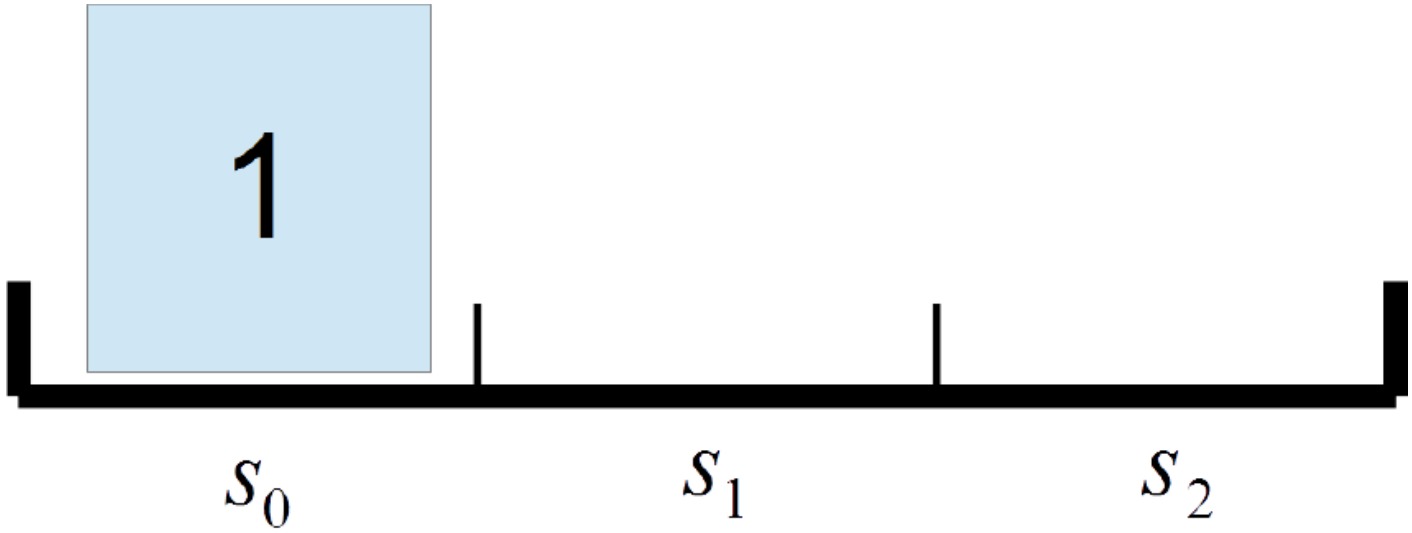
- pixelenként számol
- amit lehet sugárral metszeni, az használható
- van tükröződés, fénytörés, vetett árnyékok
- takarási feladat triviális
- sok pixel, sok sugár miatt nagy számításigény

## Inkrementális képszintézis

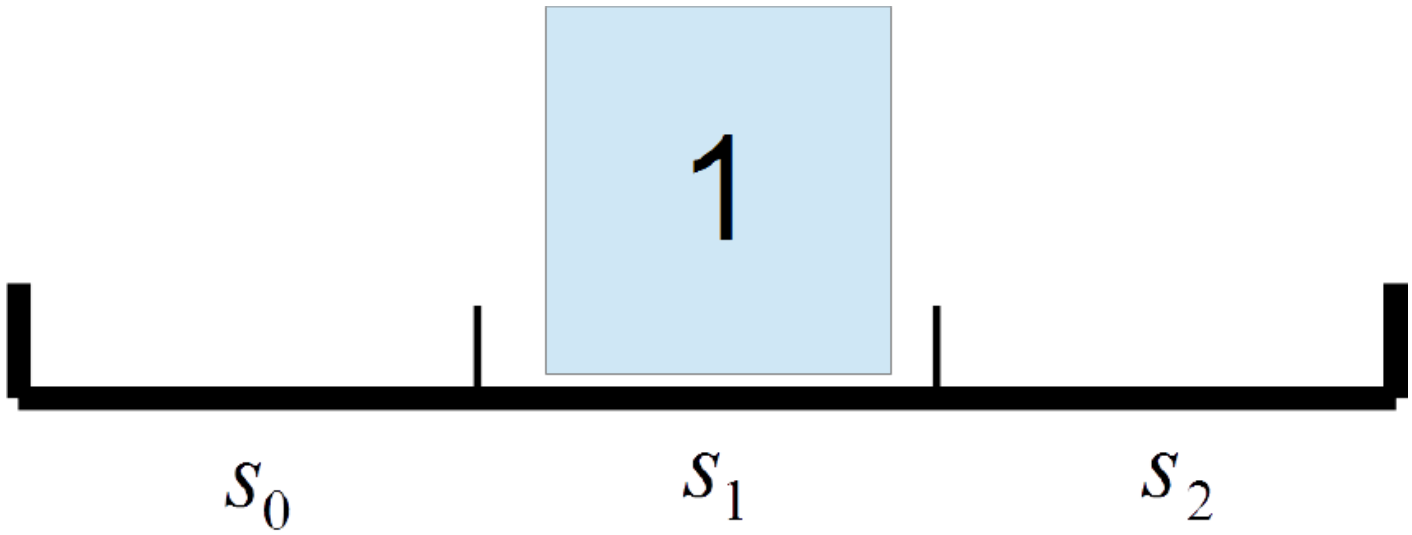
- primitívenként számol
- ami nem primitív, azt azzal kell közelíteni
- külön algoritmus kell ezekhez
- külön meg kell oldani
- a koherencia miatt kisebb számításigény

# Tartalom

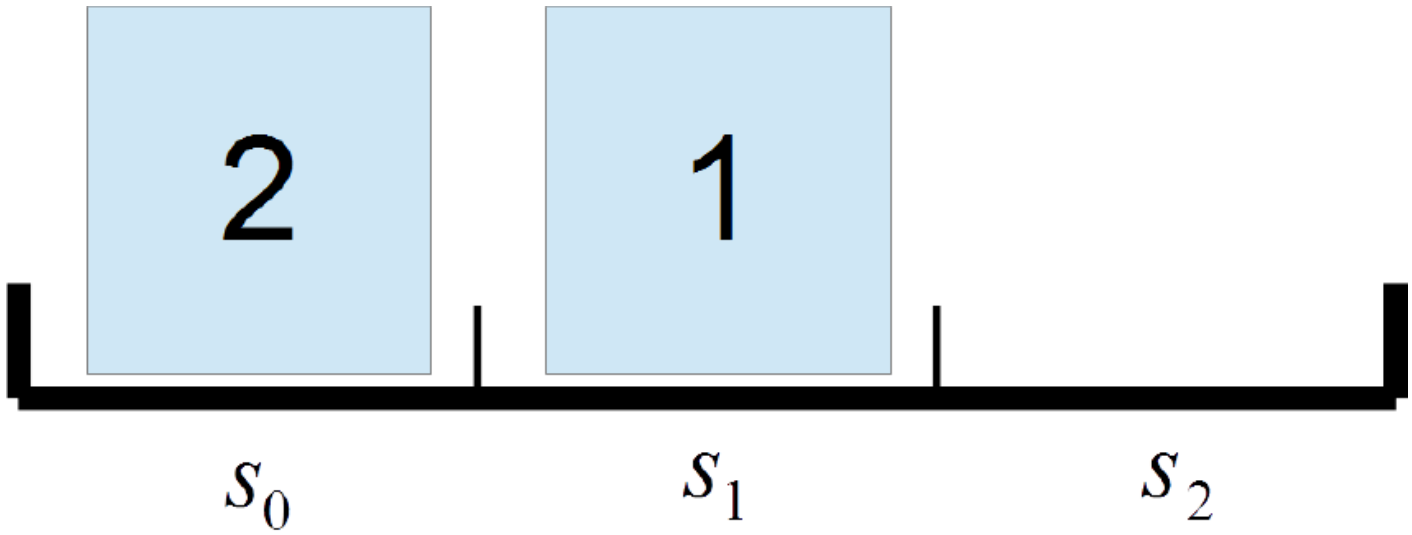
# Pipeline



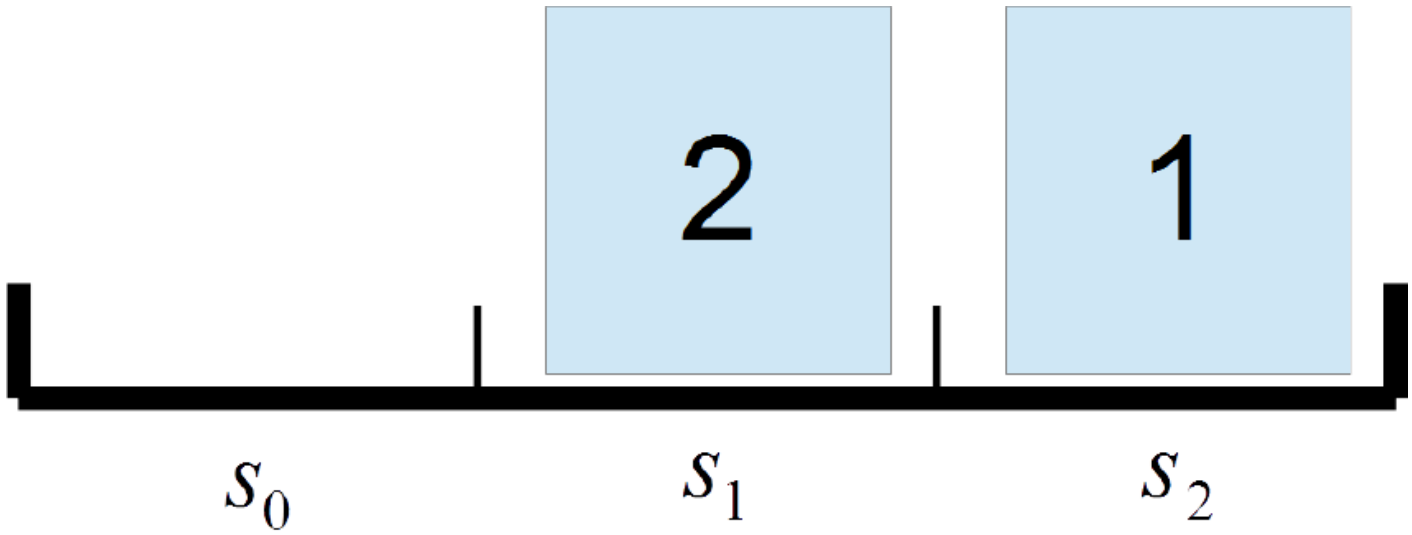
# Pipeline



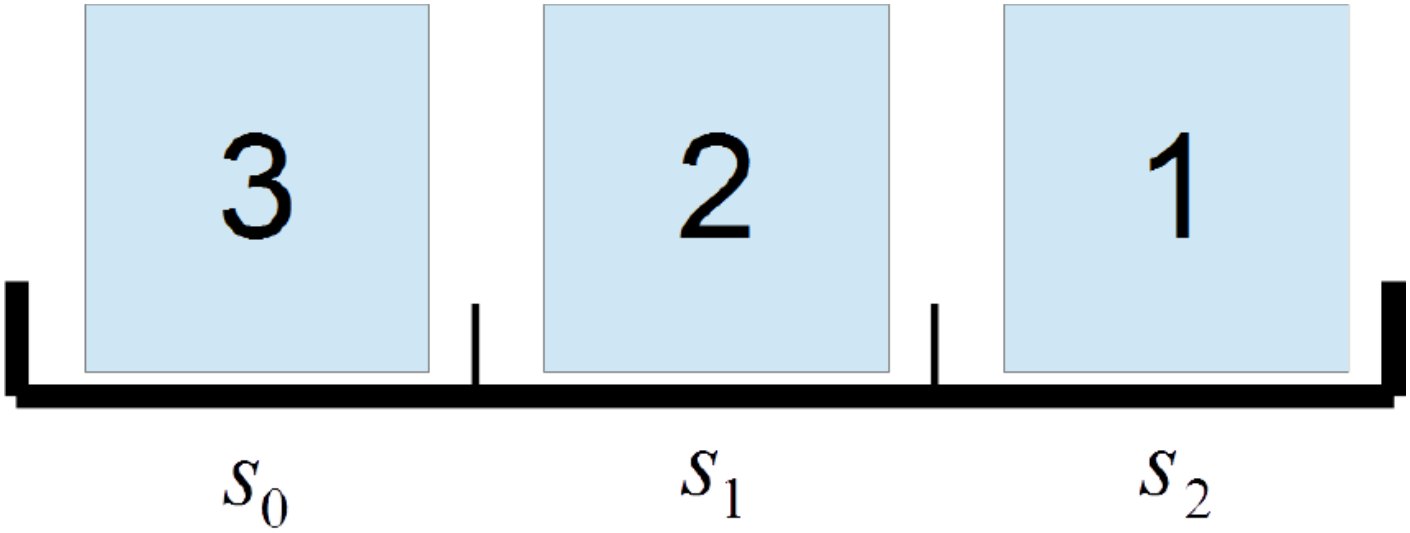
# Pipeline



# Pipeline



# Pipeline



# Pipeline

- A pipeline (vagy **szerelőszalag**, mások szerint *csővezeték*) feldolgozó egységek lánca

# Pipeline

- A pipeline (vagy **szerelőszalag**, mások szerint *csővezeték*) feldolgozó egységek lánca
- Az $s_i$ feldolgozóegység bemenete az $s_{i-1}$ feldolgozóegység kimenete

# Pipeline

- A pipeline (vagy **szerelőszalag**, mások szerint *csővezeték*) feldolgozó egységek lánca
- Az $s_i$ feldolgozóegység bemenete az $s_{i-1}$ feldolgozóegység kimenete
- Az $s_i$ kimenete pedig az $s_{i+1}$ bemenete

# Pipeline

- A pipeline (vagy **szerelőszalag**, mások szerint *csővezeték*) feldolgozó egységek lánca
- Az $s_i$ feldolgozóegység bemenete az $s_{i-1}$ feldolgozóegység kimenete
- Az $s_i$ kimenete pedig az $s_{i+1}$ bemenete
- Ha egy problémát fel tudunk osztani $n$ db egymást követő részfeladatra, amely részfeladatok nagyjából ugyanannyi idő alatt elvégezhetőek, akkor egységnyi idő alatt egyszerre $n$ munkadarabon tudunk dolgozni

# Pipeline

- A pipeline (vagy **szerelőszalag**, mások szerint *csővezeték*) feldolgozó egységek lánca
- Az $s_i$ feldolgozóegység bemenete az $s_{i-1}$ feldolgozóegység kimenete
- Az $s_i$ kimenete pedig az $s_{i+1}$ bemenete
- Ha egy problémát fel tudunk osztani $n$ db egymást követő részfeladatra, amely részfeladatok nagyjából ugyanannyi idő alatt elvégezhetőek, akkor egységnyi idő alatt egyszerre $n$ munkadarabon tudunk dolgozni
- A kezdeti felfutás és az utolsó munkadarab szalagra helyezésével induló végső kifutást leszámítva

# Grafikus szerelőszalag

- A színterünkről készített kép elkészítésének (szerelőszalagba szervezett) műveletsorozatát nevezik **grafikus szerelőszalagnak** (angolul *graphics pipeline*)

# Grafikus szerelőszalag

- A színterünkről készített kép elkészítésének (szerelőszalagba szervezett) műveletsorozatát nevezik **grafikus szerelőszalagnak** (angolul *graphics pipeline*)
- A valósidejű alkalmazásoknak lényegében a színterünk leírását kell csak átadnia, a képszintézis lépéseit a grafikus szerelőszalag végzi

# Grafikus szerelőszalag

- A színterünkről készített kép elkészítésének (szerelőszalagba szervezett) műveletsorozatát nevezik **grafikus szerelőszalagnak** (angolul *graphics pipeline*)
- A valósidejű alkalmazásoknak lényegében a színterünk leírását kell csak átadnia, a képszintézis lépéseit a grafikus szerelőszalag végzi
- A szerelőszalagban több koordinátarendszer-váltás is történik – mindent feladatot a hozzá legjobban illeszkedő rendszerben próbálunk elvégezni

# A szerelőszalag működése

- Minden primitív kirajzolását indító parancsra elindul (pl. glDrawArrays)

# A szerelőszalag működése

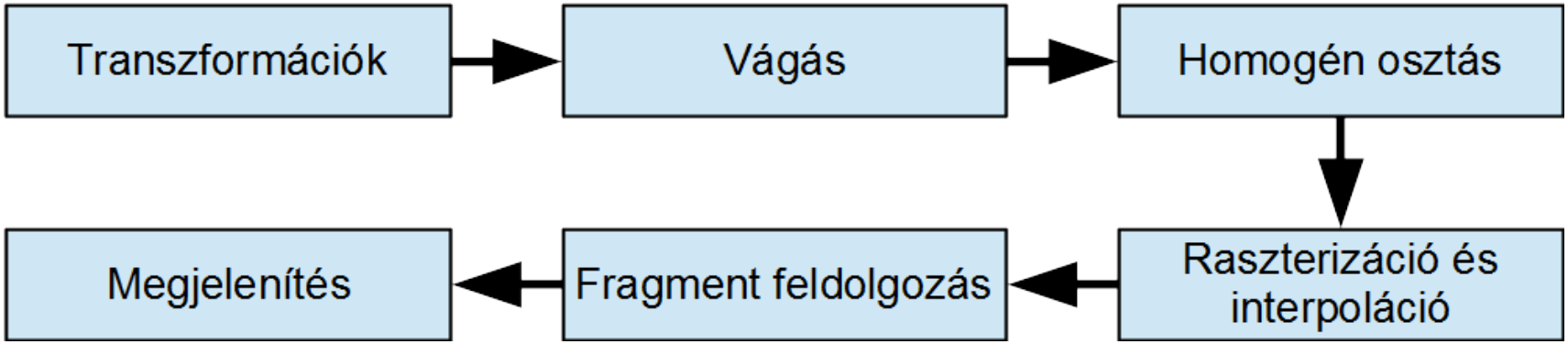
- Minden primitív kirajzolását indító parancsra elindul (pl. glDrawArrays)
- A kirajzolandó primitívek egymástól függetlenül (általában egymással párhuzamos) mennek végig a szerelőszalag összes lépésén

# A szerelőszalag működése

- Minden primitív kirajzolását indító parancsra elindul (pl. glDrawArrays)
- A kirajzolandó primitívek egymástól függetlenül (általában egymással párhuzamos) mennek végig a szerelőszalag összes lépésén
- Többféleképpen megadható és csoportosítható

# Grafikus szerelőszalag

# Grafikus szerelőszalag



# Grafikus szerelőszalag

- Lépései főbb műveletek szerint:

# Grafikus szerelőszalag

- Lépései főbb műveletek szerint:
  - Transzformációk

# Grafikus szerelőszalag

- Lépései főbb műveletek szerint:
  - Transzformációk
  - Vágás

# Grafikus szerelőszalag

- Lépései főbb műveletek szerint:
  - Transzformációk
  - Vágás
  - Homogén osztás

# Grafikus szerelőszalag

- Lépései főbb műveletek szerint:
  - Transzformációk
  - Vágás
  - Homogén osztás
  - Raszterizáció és interpoláció

# Grafikus szerelőszalag

- Lépései főbb műveletek szerint:
  - Transzformációk
  - Vágás
  - Homogén osztás
  - Raszterizáció és interpoláció
  - Fragment feldolgozás

# Grafikus szerelőszalag

- Lépései főbb műveletek szerint:
  - Transzformációk
  - Vágás
  - Homogén osztás
  - Raszterizáció és interpoláció
  - Fragment feldolgozás
  - Megjelenítés

# Grafikus szerelőszalag

- Lépései főbb műveletek szerint:
  - Transzformációk
  - Vágás
  - Homogén osztás
  - Raszterizáció és interpoláció
  - Fragment feldolgozás
  - Megjelenítés
- A grafikus szerelőszalag eredménye egy kép (egy kétdimenziós pixeltömb, aminek minden elemében egy színérték található)

# A szerelőszalag bemeneti adatai

- Ábrázolandó tárgyak *geometriai* és *optikai modellje*

# A szerelőszalag bemeneti adatai

- Ábrázolandó tárgyak *geometriai* és *optikai modellje*
- *Virtuális kamera* adatai (nézőpont és látógúla)

# A szerelőszalag bemeneti adatai

- Ábrázolandó tárgyak *geometriai* és *optikai modellje*
- *Virtuális kamera* adatai (nézőpont és látógúla)
- A képkeret (az a pixeltömb, amire a színterünk síkvetületét leképezzük)

# A szerelőszalag bemeneti adatai

- Ábrázolandó tárgyak *geometriai* és *optikai modellje*
- *Virtuális kamera* adatai (nézőpont és látógúla)
- A képkeret (az a pixeltömb, amire a színterünk síkvetületét leképezzük)
- A színtérben található fényforrásokhoz és anyagokhoz tartozó *megvilágítási adatok*

# Tartalom

# Transzformációk

- A szerelőszalag transzformációinak feladata: a modelltérben adott objektumot „eljuttatni” a képernyő-térbe

# Transzformációk

- A szerelőszalag transzformációinak feladata: a modelltérben adott objektumot „eljuttatni” a képernyő-térbe
- Lépései (összesítve):

# Transzformációk

- A szerelőszalag transzformációinak feladata: a modelltérben adott objektumot „eljuttatni” a képernyő-térbe
- Lépései (összesítve):

modell k.r.

# Transzformációk

- A szerelőszalag transzformációinak feladata: a modelltérben adott objektumot „eljuttatni” a képernyő-térbe
- Lépései (összesítve):

modell k.r.

$\rightarrow$ világ k.r.

# Transzformációk

- A szerelőszalag transzformációinak feladata: a modelltérben adott objektumot „eljuttatni” a képernyő-térbe
- Lépései (összesítve):

modell k.r.
$\rightarrow$ világ k.r.
$\rightarrow$ kamera k.r.

# Transzformációk

- A szerelőszalag transzformációinak feladata: a modelltérben adott objektumot „eljuttatni” a képernyő-térbe
- Lépései (összesítve):

modell k.r.
$\rightarrow$ világ k.r.
$\rightarrow$ kamera k.r.
$\rightarrow$ normalizált eszköz k.r.

# Transzformációk

- A szerelőszalag transzformációinak feladata: a modelltérben adott objektumot „eljuttatni” a képernyő-térbe
- Lépései (összesítve):

modell k.r.
$\rightarrow$ világ k.r.
$\rightarrow$ kamera k.r.
$\rightarrow$ normalizált eszköz k.r.
$\rightarrow$ képernyő k.r.

# Transzformációk

- A primitívek olyanok, hogy elegendő (*) a csúcspontjaikat transzformálni és utána a transzformált csúcspontokat összekötni

# Transzformációk

- A primitívek olyanok, hogy elegendő (*) a csúcspontjaikat transzformálni és utána a transzformált csúcspontokat összekötni
- Ezért a transzformációkat a primitívek csúcspontjain végezzük el

# Transzformációk

- A primitívek olyanok, hogy elegendő (*) a csúcspontjaikat transzformálni és utána a transzformált csúcspontokat összekötni
- Ezért a transzformációkat a primitívek csúcspontjain végezzük el
- (*): ez középpontos vetítésnél óvatosan kezelendő!

# Transzformációk

- A vágás előtt nem végzünk homogén osztást

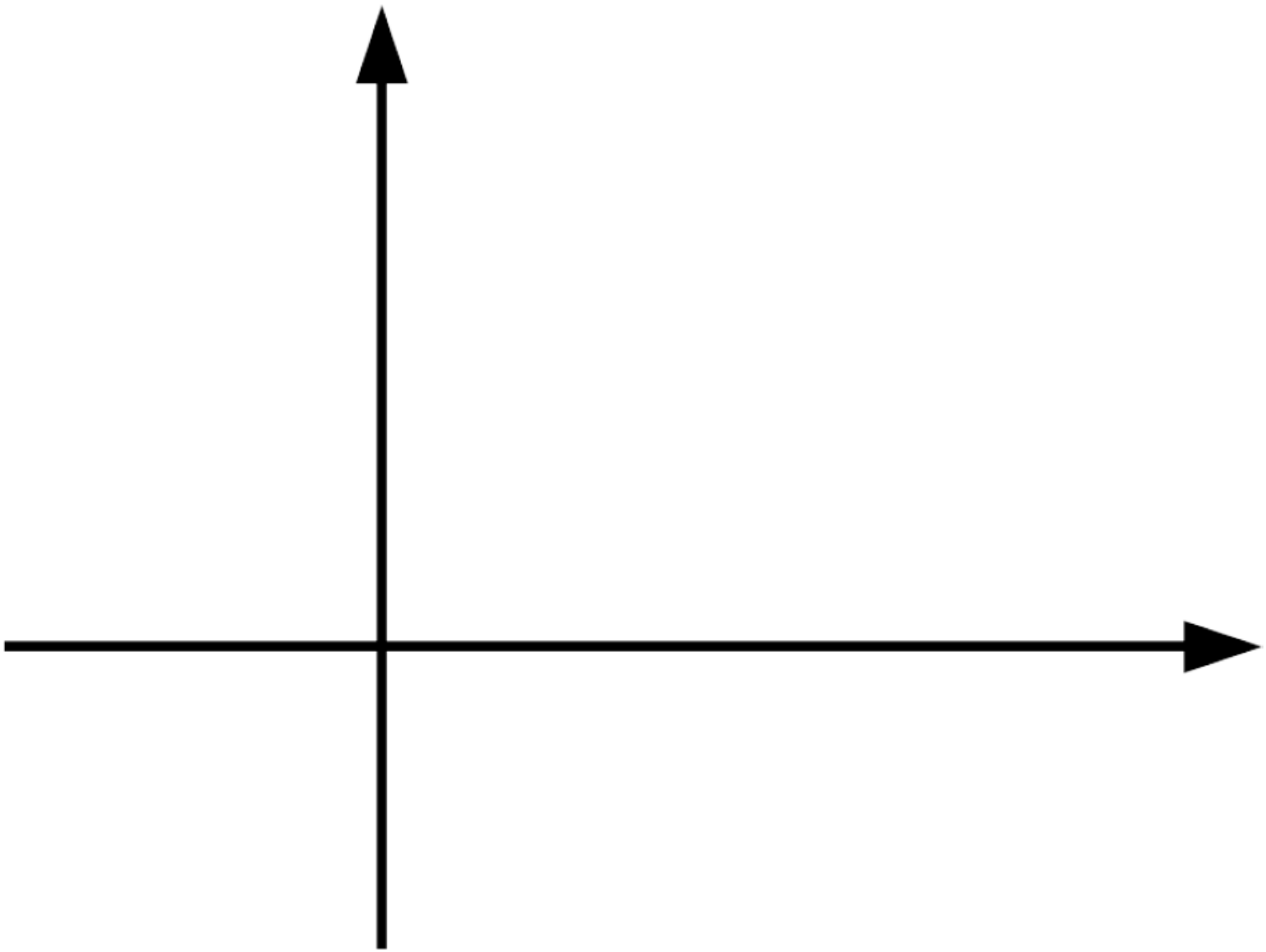
# Transzformációk

- A vágás előtt nem végzünk homogén osztást
- Tehát a Transzformációk nevű fázisa a szerelőszalagnak nem megy végig a képernyő KR-ig, megáll a homogén osztás **előtti**, de középpontos vetítés utáni homogén térben

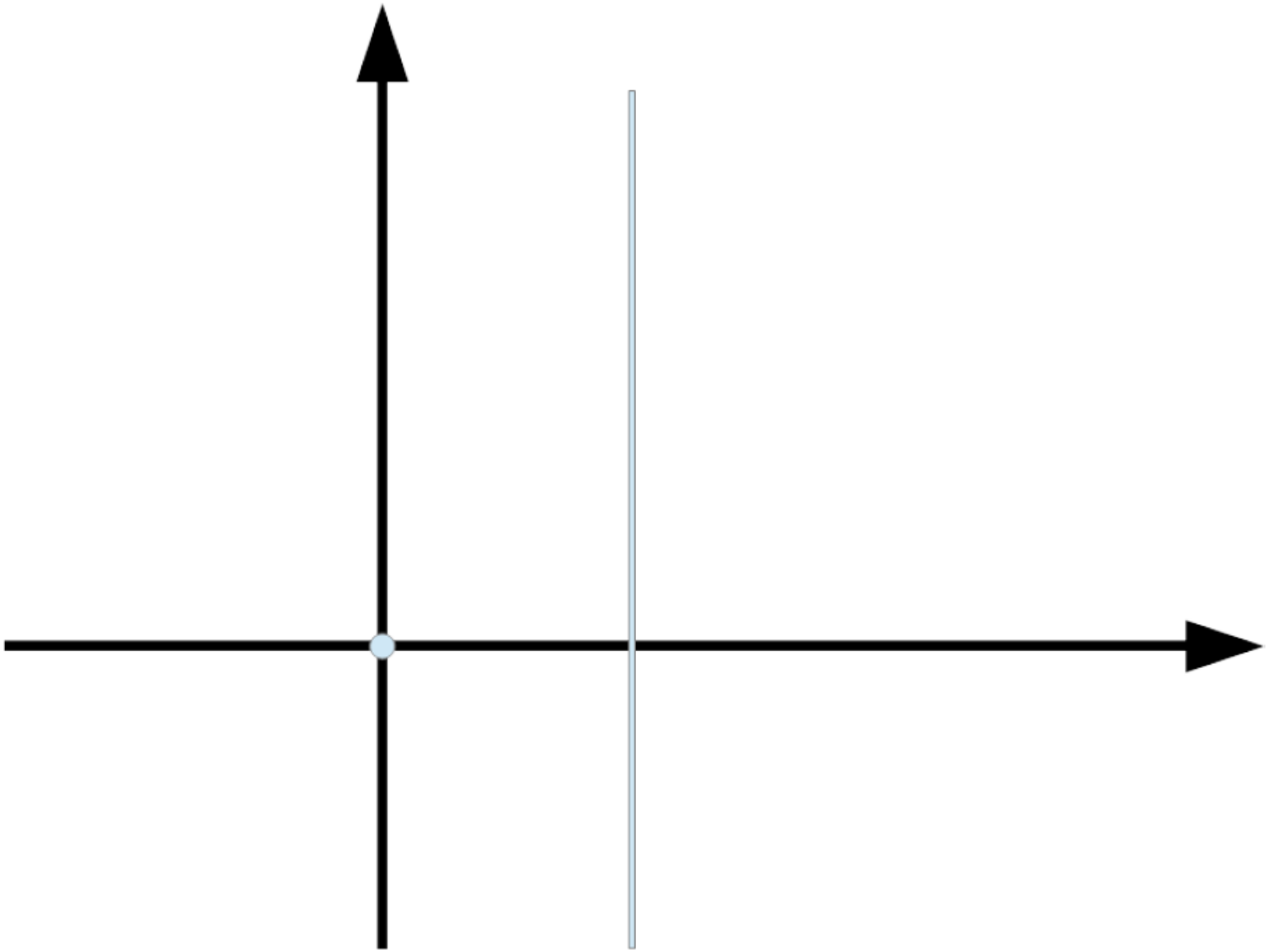
# Transzformációk

- A vágás előtt nem végzünk homogén osztást
- Tehát a Transzformációk nevű fázisa a szerelőszalagnak nem megy végig a képernyő KR-ig, megáll a homogén osztás **előtti**, de középpontos vetítés utáni homogén térben
- Gyakorlaton ezt írjuk bele a vertex shaderben a gl_Position változóba

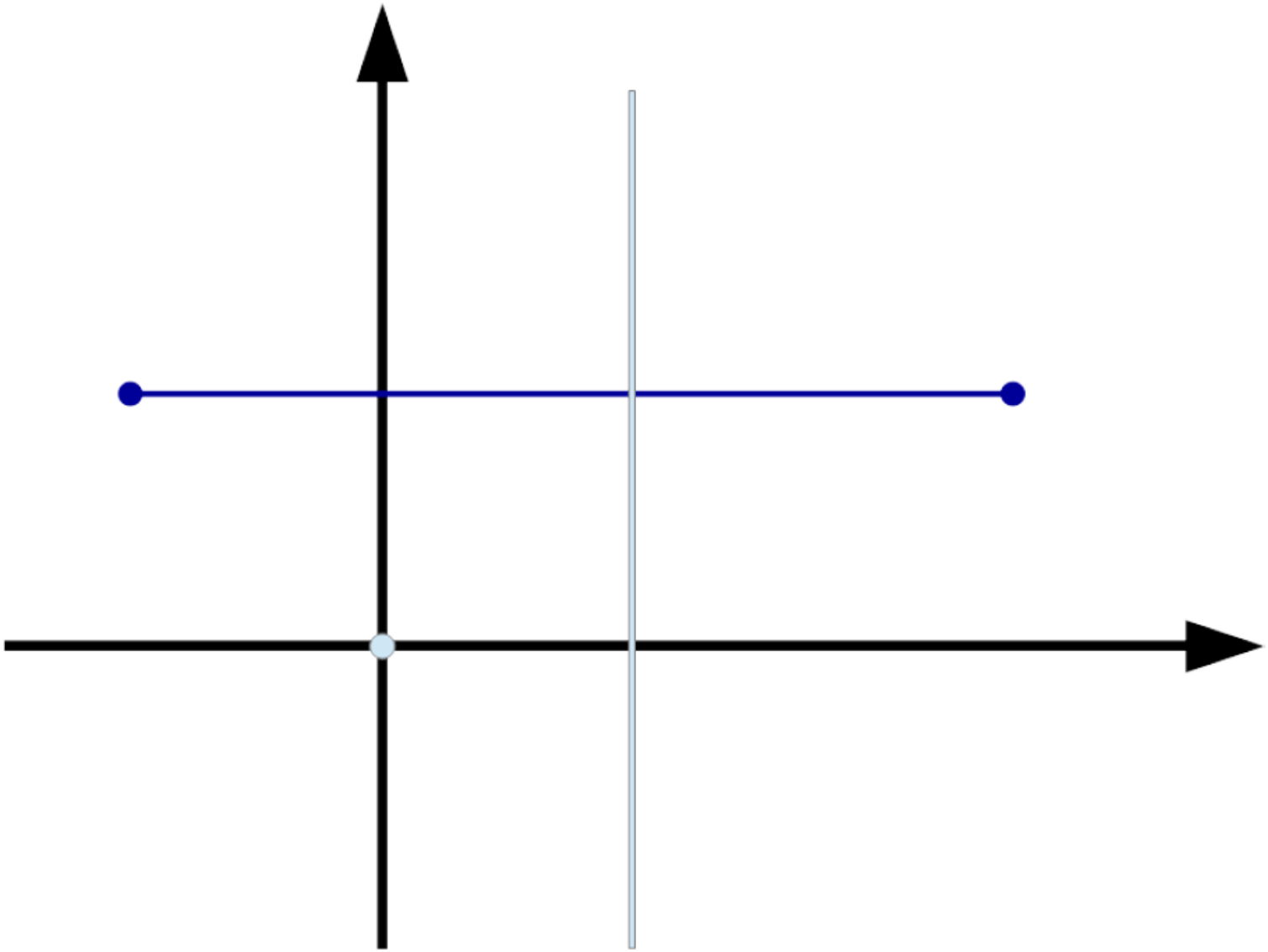
# Átfordulási probléma



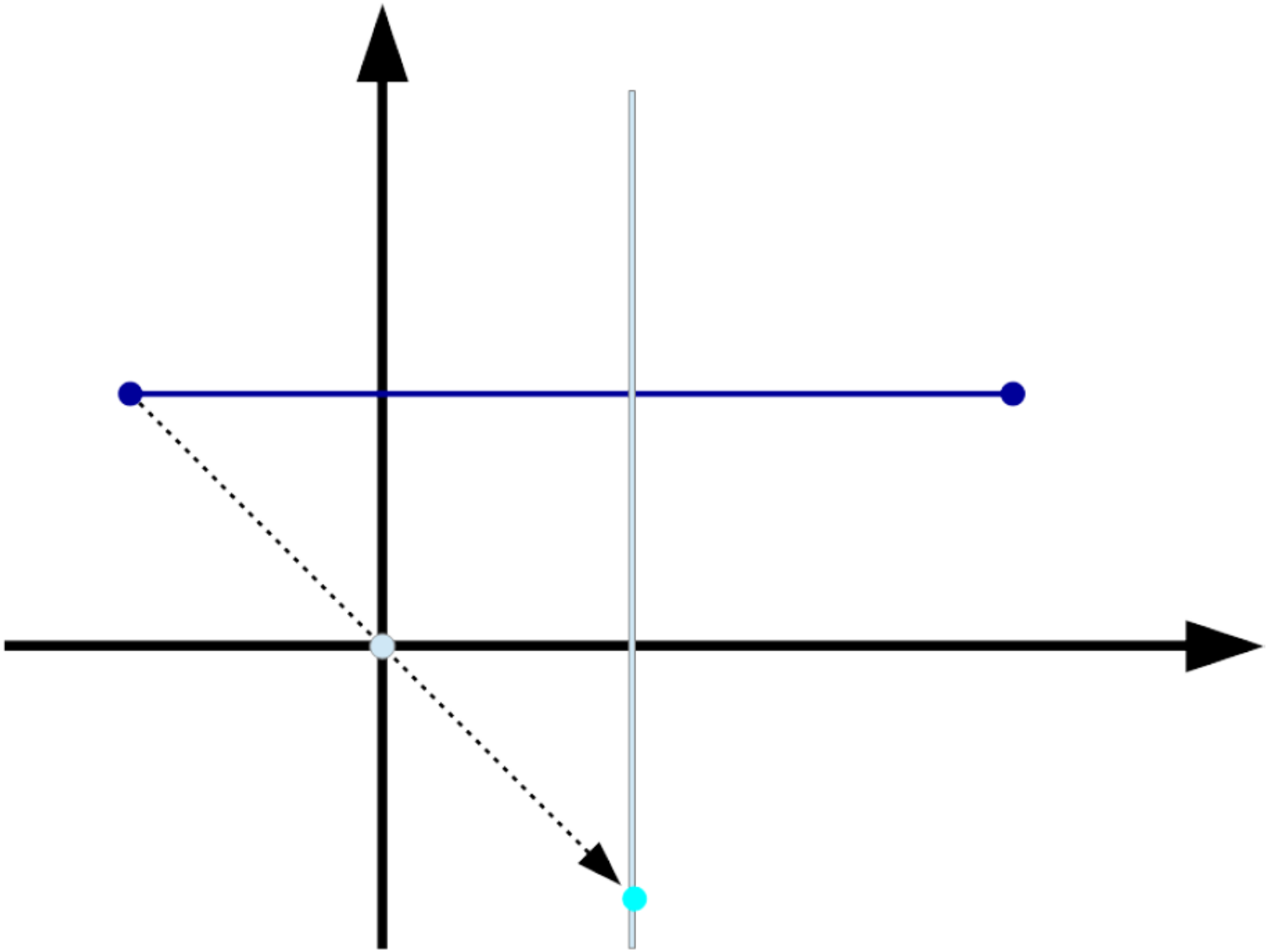
# Átfordulási probléma



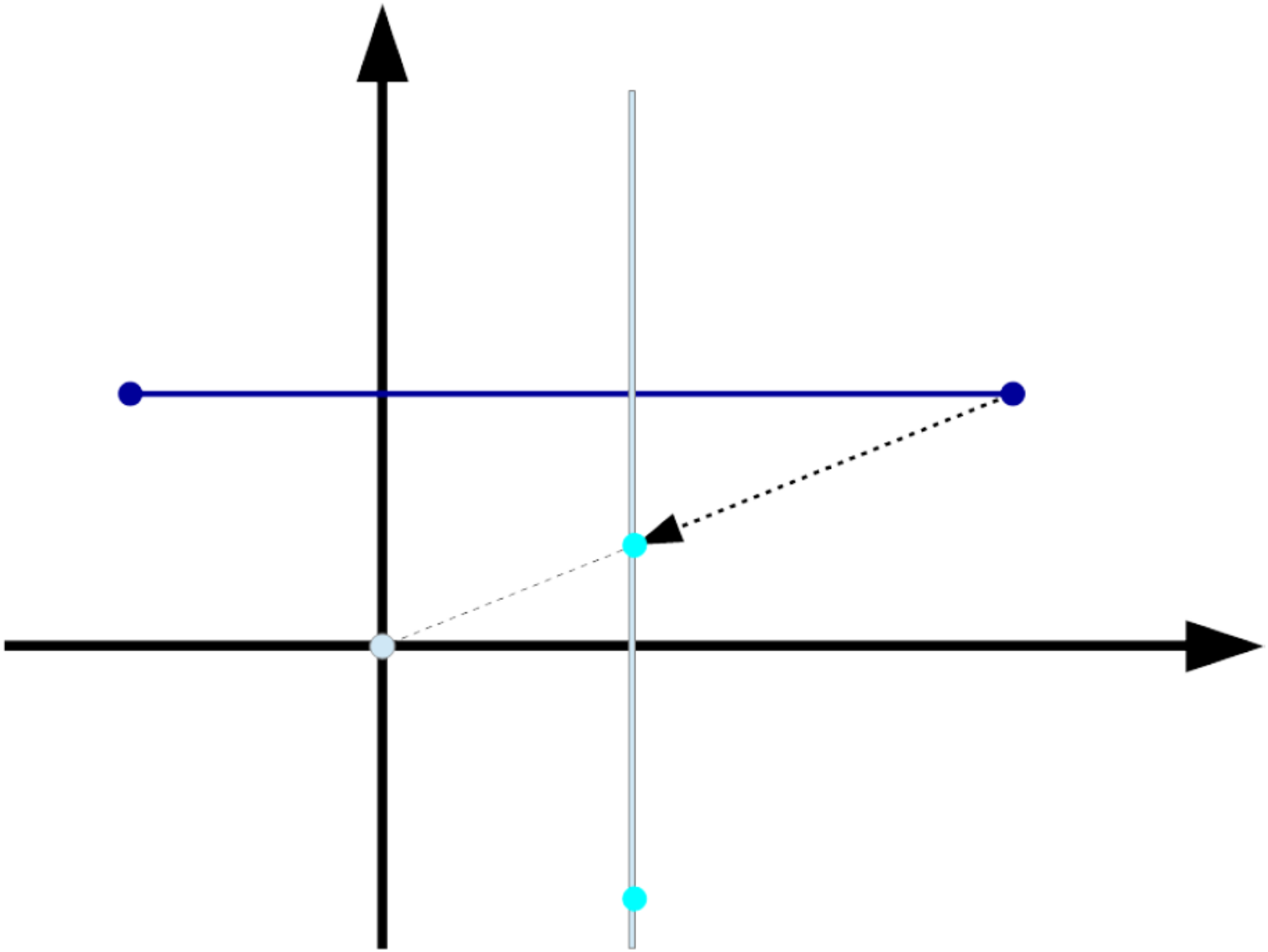
# Átfordulási probléma



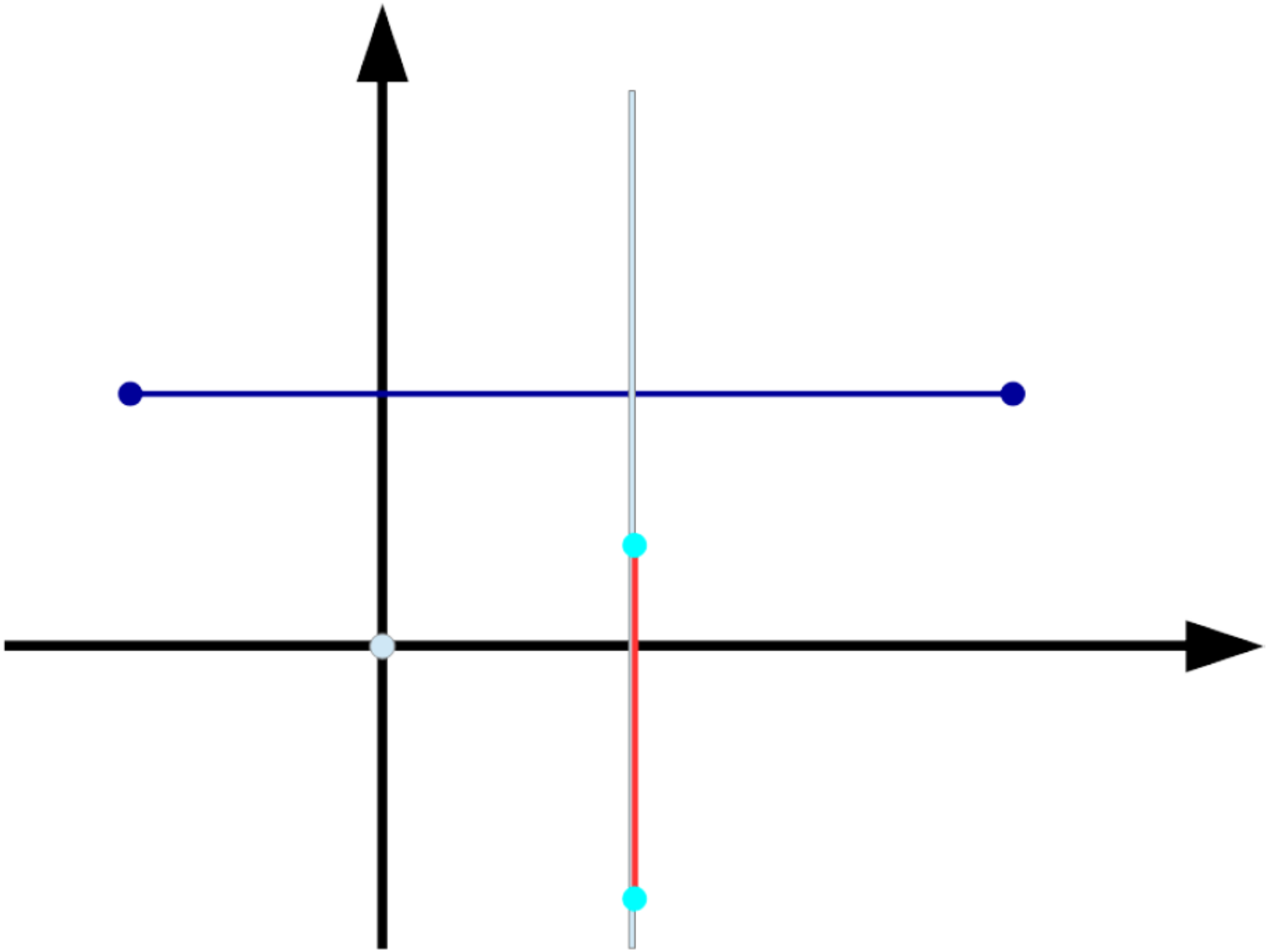
# Átfordulási probléma



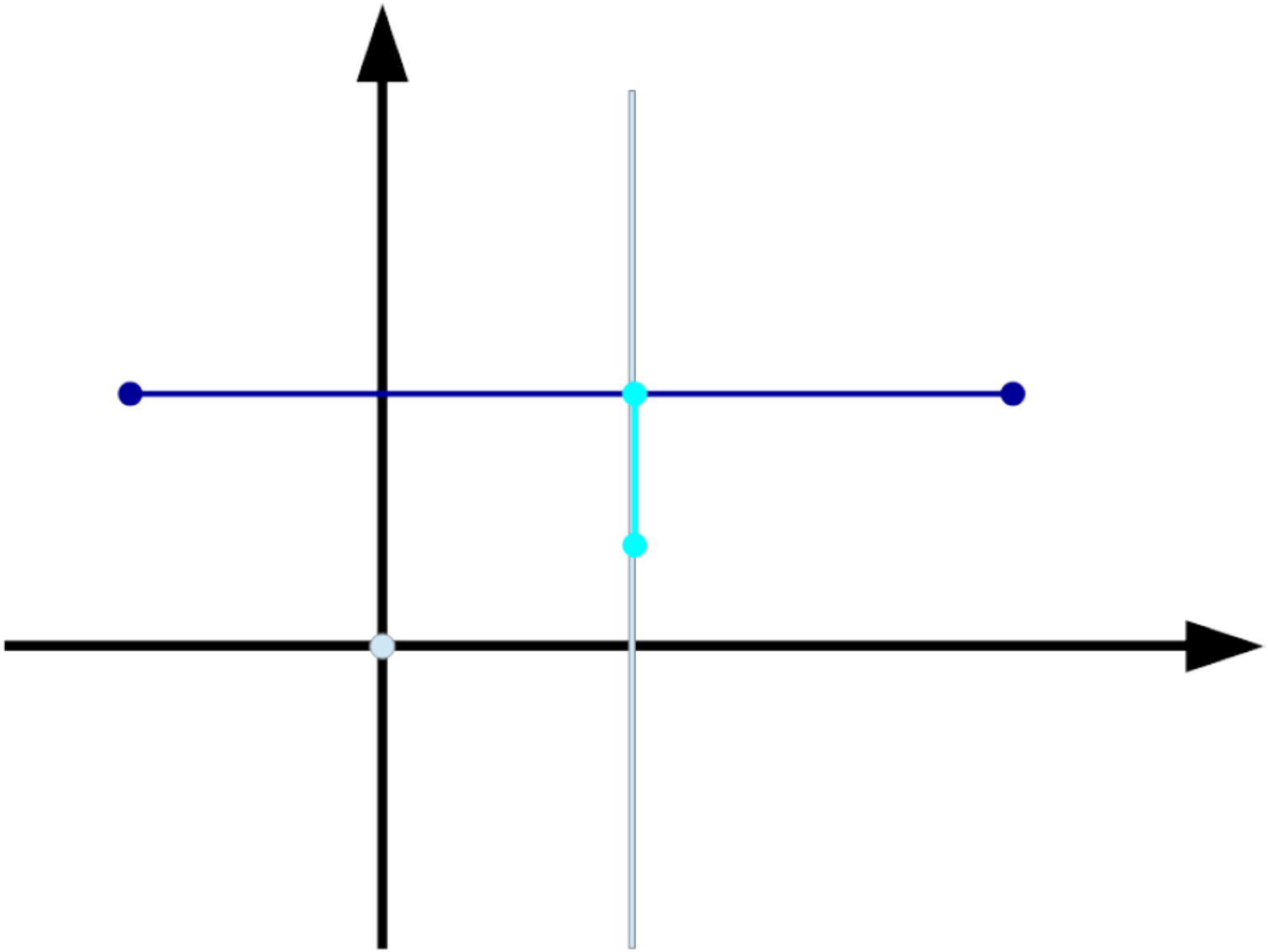
# Átfordulási probléma



# Átfordulási probléma



# Átfordulási probléma



# Koordináta-rendszerek

- Normalizált eszköz KR:
  A hardverre jellemző, $[-1,1] \times [-1,1] \times [-1,1]$ vagy $[-1,1] \times [-1,1] \times [0,1]$ kiterjedésű KR.

# Koordináta-rendszerek

- Normalizált eszköz KR:
  A hardverre jellemző, $[-1,1] \times [-1,1] \times [-1,1]$ vagy $[-1,1] \times [-1,1] \times [0,1]$ kiterjedésű KR.
- Képernyő KR:
  A megjelenítendő képnek (képernyő/ablak) megfelelő KR (balkezes, bal-felső „sarok” az origó).

# Modellezési (világ) transzformáció

- A saját (modell) koordináta-rendszerben adott modelleket a világ-koordináta rendszerben helyezi el

# Modellezési (világ) transzformáció

- A saját (modell) koordináta-rendszerben adott modelleket a világ-koordináta rendszerben helyezi el
- Tipikusan minden modellre különböző (lehet a színterünk két elem csak a világtrafóban különbözik!)

# Modellezési (világ) transzformáció

- A saját (modell) koordináta-rendszerben adott modelleket a világ-koordináta rendszerben helyezi el
- Tipikusan minden modellre különböző (lehet a színterünk két elem csak a világtrafóban különbözik!)
- Jellemzően affin transzformációk

# Modellezési (világ) transzformáció

- A saját (modell) koordináta-rendszerben adott modelleket a világ-koordináta rendszerben helyezi el
- Tipikusan minden modellre különböző (lehet a színterünk két elem csak a világtrafóban különbözik!)
- Jellemzően affin transzformációk
- Gyakorlatban: ez a *model* (vagy *world*) mátrix a kódjainkban

# Nézeti (kamera) transzformáció

- A világ koordináta-rendszert a kamerához rögzített koordináta-rendszerbe viszi át.

# Nézeti (kamera) transzformáció

- A világ koordináta-rendszert a kamerához rögzített koordináta-rendszerbe viszi át.
- A transzformáció a kamera tulajdonságaiból adódik.

# Nézeti (kamera) transzformáció

- A világ koordináta-rendszert a kamerához rögzített koordináta-rendszerbe viszi át.
- A transzformáció a kamera tulajdonságaiból adódik.
- Gyakorlatban: ez a *view* vagy *camera* mátrix.

# Nézeti (kamera) transzformáció

- Tulajdonságok ugyanazok, mint a sugárkövetés esetén: **eye**, **center**, **up**

# Nézeti (kamera) transzformáció

- Tulajdonságok ugyanazok, mint a sugárkövetés esetén: **eye**, **center**, **up**
- Ebből kapjuk a nézeti koordináta-rendszer tengelyeit:

$$\mathbf{w} = \frac{\mathbf{eye} - \mathbf{center}}{|\mathbf{eye} - \mathbf{center}|}$$

$$\mathbf{u} = \frac{\mathbf{up} \times \mathbf{w}}{|\mathbf{up} \times \mathbf{w}|}$$

$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$

# Nézeti(kamera) transzformáció mátrixa

- Áttérés az **eye** origójú, $\mathbf{u}, \mathbf{v}, \mathbf{w}$ koordináta-rendszerbe:

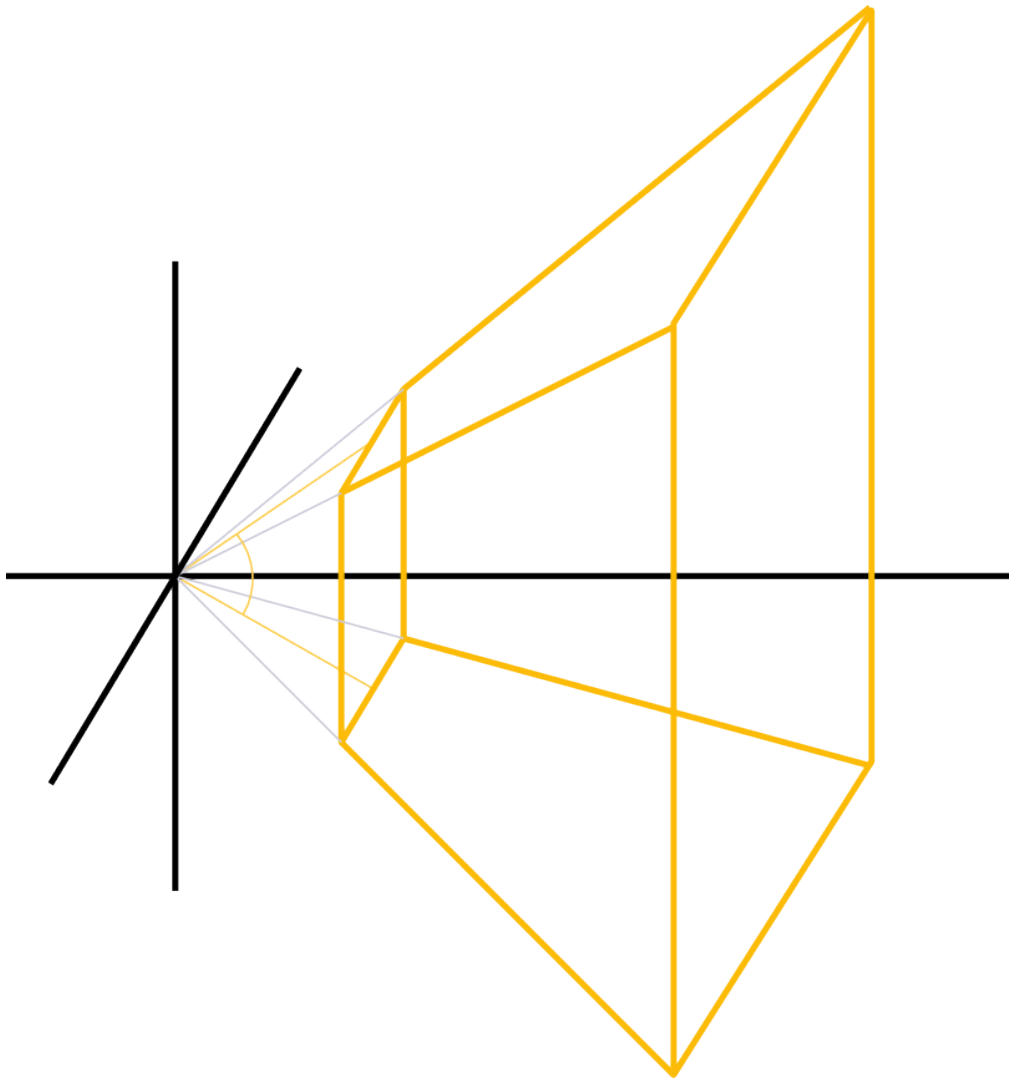
$$T_{View} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Vetítés – párhuzamos vetítés

- A mátrix ami megadja egyszerű, például az $XY$ síkra való vetítés

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Vetítés – középpontos vetítés



# Vetítés – középpontos vetítés

- Emlékeztető: 3. EA

# Vetítés – középpontos vetítés

- Emlékeztető: 3. EA
- A nézeti csonka gúla által határolt térrészt normalizált eszköz KR-be viszi át

# Vetítés – középpontos vetítés

- Emlékeztető: 3. EA
- A nézeti csonka gúla által határolt térrészt normalizált eszköz KR-be viszi át
- Ami benne volt a csonka gúlában, az lesz benne a $[-1,1] \times [-1,1] \times [0,1]$ (vagy $[-1,1] \times [-1,1] \times [-1,1]$) tartományban

# Vetítés – középpontos vetítés

- Emlékeztető: 3. EA
- A nézeti csonka gúla által határolt térrészt normalizált eszköz KR-be viszi át
- Ami benne volt a csonka gúlában, az lesz benne a $[-1,1] \times [-1,1] \times [0,1]$ (vagy $[-1,1] \times [-1,1] \times [-1,1]$) tartományban
- A transzformáció a kamerán átmenő *vetítő sugarakból* párhuzamosokat csinál

# Vetítés – középpontos vetítés

- Emlékeztető: 3. EA
- A nézeti csonka gúla által határolt térrészt normalizált eszköz KR-be viszi át
- Ami benne volt a csonka gúlában, az lesz benne a $[-1,1] \times [-1,1] \times [0,1]$ (vagy $[-1,1] \times [-1,1] \times [-1,1]$) tartományban
- A transzformáció a kamerán átmenő *vetítő sugarakból* párhuzamosokat csinál
- A transzformáció a kamerapozíciót a végtelenbe tolja

# Vetítés – középpontos vetítés

- Emlékeztető: 3. EA
- A nézeti csonka gúla által határolt térrészt normalizált eszköz KR-be viszi át
- Ami benne volt a csonka gúlában, az lesz benne a $[-1,1] \times [-1,1] \times [0,1]$ (vagy $[-1,1] \times [-1,1] \times [-1,1]$) tartományban
- A transzformáció a kamerán átmenő *vetítő sugarakból* párhuzamosokat csinál
- A transzformáció a kamerapozíciót a végtelenbe tolja
- Gyakorlatban: ez a *proj* mátrix

# Projektív transzformáció

- Emlékeztető: tulajdonságok
  - függőleges és vízszintes nyílásszög *(fovx, fovy)* vagy az alap oldalainak az aránya és a függőleges nyílásszög *(fovy, aspect)*,
  - a közeli vágósík távolsága *(near)*,
  - a távoli vágósík távolsága *(far)*

# Projektív transzformáció a szerelőszalagban

1. Normalizáljuk a látógúlát, hogy mindkét nyílásszöge $\frac{\pi}{2}$ legyen

# Projektív transzformáció a szerelőszalagban

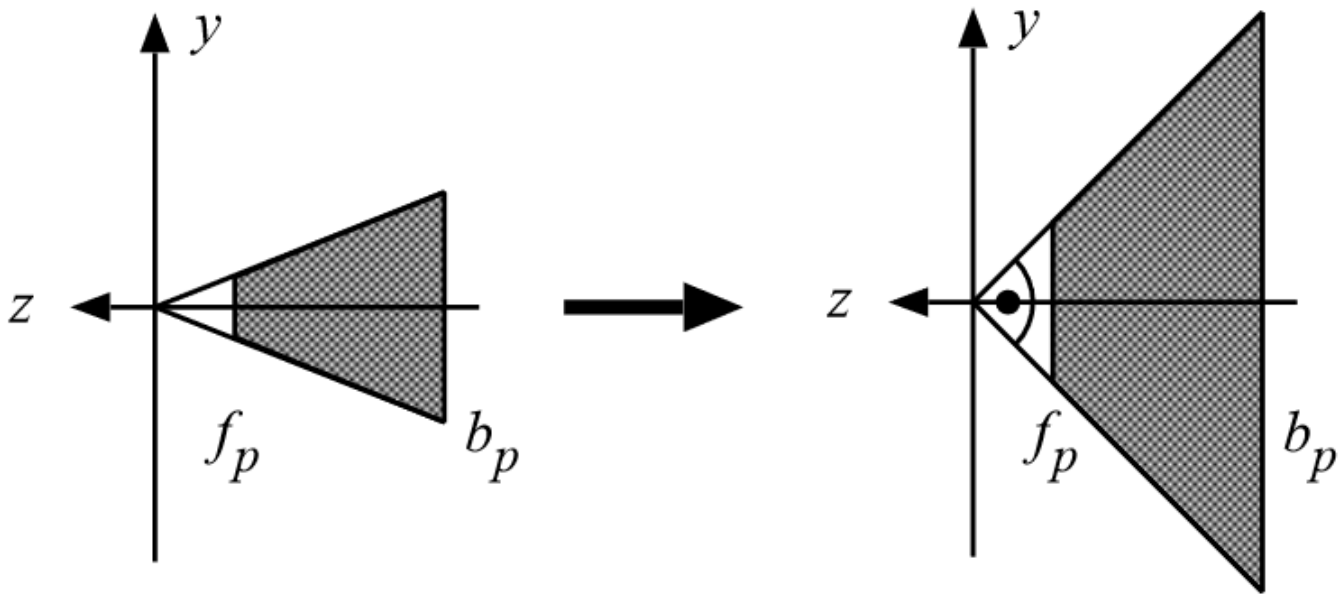
1. Normalizáljuk a látógúlát, hogy mindkét nyílásszöge $\frac{\pi}{2}$ legyen
2. Végezzük el a középpontos vetítést (az origóból, az origótól 1 távolságra lévő, $XY$ síkkal párhuzamos síkra)

# Projektív transzformáció a szerelőszalagban

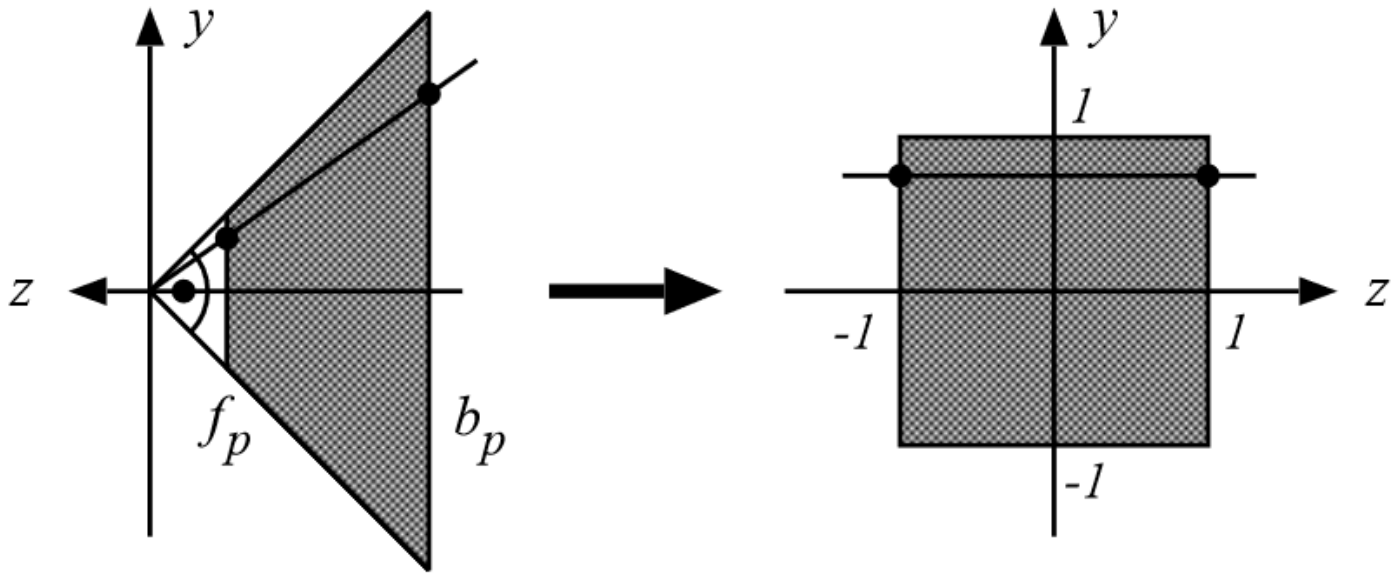
1. Normalizáljuk a látógúlát, hogy mindkét nyílásszöge $\frac{\pi}{2}$ legyen
2. Végezzük el a középpontos vetítést (az origóból, az origótól 1 távolságra lévő, $XY$ síkkal párhuzamos síkra)
3. Képezzük át rendre a $z = near$, $z = far$ síkokat a $z = -1$ és $z = 1$ síkokra

Így (1)–(2)-vel az eredmény koordinátákat $x$ és $y$ szerint normalizáljuk (képezzük bele $[-1, 1]$-be), a (3)-mal pedig a $z$ szerint

# Látógúla normalizálása (1)



# Látógúla normalizálása (2-3)



# Látógúla normalizálása (1)

- Figyeljünk: a szerelőszalagunk ezen pontján a kamera $-Z$ felé néz és az origóban van

# Látógúla normalizálása (1)

- Figyeljünk: a szerelőszalagunk ezen pontján a kamera $-Z$ felé néz és az origóban van
- A fenti térből térjünk át egy „normalizáltabb” gúlába – aminek a nyílásszöge $x$ és $y$ mentén is 90 fokos

# Látógúla normalizálása (1)

- Mátrix alakban:

$$\begin{bmatrix} 1/\tan\frac{fovx}{2} & 0 & 0 & 0 \\ 0 & 1/\tan\frac{fovy}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Középpontos vetítés (2)

- Az origó, mint vetítési középpont és egy, attól a $Z$ tengely mentén $d$ egységre található, $XY$ síkkal párhuzamos vetítősíkra való vetítés mátrixa:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix}$$

# Mélység normalizálás (3)

- Ezután már csak a közeli és a távoli vágósík $z$ koordinátáit kell a normalizálásnak megfelelően átképezni ($-1, 1$ vagy $0, 1$-re):

$$T_{Projection} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{far}{near-far} & \frac{near*far}{near-far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

# Tartalom

# Vágás

- Cél: ne dolgozzunk feleslegesen azokkal az elemekkel, amikből nem lesz pixel (nem jelennek majd meg)

# Vágás

- Cél: ne dolgozzunk feleslegesen azokkal az elemekkel, amikből nem lesz pixel (nem jelennek majd meg)
- Ehhez szűrjük ki azokat az elemeket, amik *biztosan* nem kerülhetnek a képernyőre

# Vágás

- Cél: ne dolgozzunk feleslegesen azokkal az elemekkel, amikből nem lesz pixel (nem jelennek majd meg)
- Ehhez szűrjük ki azokat az elemeket, amik *biztosan* nem kerülhetnek a képernyőre
- Továbbá azokat, amiknek csak darabjai kerülhetnek a képernyőre *vágjuk* (azonosítsuk azon darabjait, amik teljes egészükben belelógnak a képernyőbe és fedjük le ezeket a darabokat primitívekkel)

# Vágás homogén koordinátákban

- Most csak egyetlen pont vágását tekintjük, homogén koordinátákban:

# Vágás homogén koordinátákban

- Most csak egyetlen pont vágását tekintjük, homogén koordinátákban:
- Legyen: $[x_h, y_h, z_h, h]^T = \mathbf{M}_{\text{Proj}} \cdot [x_c, y_c, z_c, 1]^T$

# Vágás homogén koordinátákban

- Most csak egyetlen pont vágását tekintjük, homogén koordinátákban:
- Legyen: $[x_h, y_h, z_h, h]^T = \mathbf{M}_{\text{Proj}} \cdot [x_c, y_c, z_c, 1]^T$
- Cél:
  $[x, y, z]^T := [x_h/h, y_h/h, z_h/h]^T \in [-1, 1] \times [-1, 1] \times [-1, 1]$,
  azaz

# Vágás homogén koordinátákban

- Most csak egyetlen pont vágását tekintjük, homogén koordinátákban:
- Legyen: $[x_h, y_h, z_h, h]^T = \mathbf{M}_{\text{Proj}} \cdot [x_c, y_c, z_c, 1]^T$
- Cél:
  $[x, y, z]^T := [x_h/h, y_h/h, z_h/h]^T \in [-1, 1] \times [-1, 1] \times [-1, 1]$,
  azaz

Legyen $h > 0$, és

$$-1 \leq x \leq 1$$
$$-1 \leq y \leq 1$$
$$-1 \leq z \leq 1$$

# Vágás homogén koordinátákban

- Most csak egyetlen pont vágását tekintjük, homogén koordinátákban:
- Legyen: $[x_h, y_h, z_h, h]^T = \mathbf{M}_{\text{Proj}} \cdot [x_c, y_c, z_c, 1]^T$
- Cél:
  $[x, y, z]^T := [x_h/h, y_h/h, z_h/h]^T \in [-1, 1] \times [-1, 1] \times [-1, 1]$,
  azaz

Legyen $h > 0$, és

$$-1 \leq x \leq 1$$
$$-1 \leq y \leq 1$$
$$-1 \leq z \leq 1$$

Ebből kapjuk:

$$-h \leq x_h \leq h$$
$$-h \leq y_h \leq h$$
$$-h \leq z_h \leq h$$

# Tartalom

# Raszterizáció

- Ne feledjük: eddig minden primitív, amiről beszéltünk folytonos objektum volt

# Raszterizáció

- Ne feledjük: eddig minden primitív, amiről beszéltünk folytonos objektum volt
- Azonban nekünk egy diszkrét térben, a képernyő képpontjain kell dolgoznunk

# Raszterizáció

- Ne feledjük: eddig minden primitív, amiről beszéltünk folytonos objektum volt
- Azonban nekünk egy diszkrét térben, a képernyő képpontjain kell dolgoznunk
- Vagyis diszkretizálni kell a folytonos primitíveinket – ezt hívják *raszterizációnak*

# Raszterizáció

- Olyan geometriai primitíveket kell választanunk, amelyeket gyorsan tudunk raszterizálni

# Raszterizáció

- Olyan geometriai primitíveket kell választanunk, amelyeket gyorsan tudunk raszterizálni
- Mi lehet ilyen? Jó lenne pl. ha egyik pixelhez tartozó felületi pontjának koordinátái alapján könnyen számítható lenne a szomszédos pixelekhez tartozó pontok koordinátái, illetve ha síkbeli is lenne...

# Raszterizáció

- Olyan geometriai primitíveket kell választanunk, amelyeket gyorsan tudunk raszterizálni
- Mi lehet ilyen? Jó lenne pl. ha egyik pixelhez tartozó felületi pontjának koordinátái alapján könnyen számítható lenne a szomszédos pixelekhez tartozó pontok koordinátái, illetve ha síkbeli is lenne...
- A háromszög ilyen!

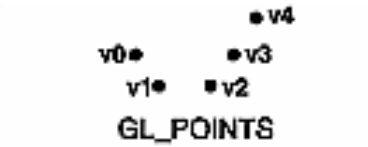
# Raszterizáció

- Olyan geometriai primitíveket kell választanunk, amelyeket gyorsan tudunk raszterizálni
- Mi lehet ilyen? Jó lenne pl. ha egyik pixelhez tartozó felületi pontjának koordinátái alapján könnyen számítható lenne a szomszédos pixelekhez tartozó pontok koordinátái, illetve ha síkbeli is lenne...
- A háromszög ilyen!
- Minden egyéb felületet ilyen primitívekkel (lényegében: síklapokkal) közelítünk

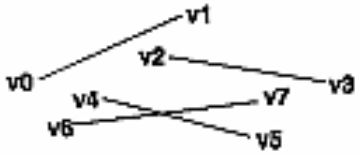
# Raszterizáció

- Olyan geometriai primitíveket kell választanunk, amelyeket gyorsan tudunk raszterizálni
- Mi lehet ilyen? Jó lenne pl. ha egyik pixelhez tartozó felületi pontjának koordinátái alapján könnyen számítható lenne a szomszédos pixelekhez tartozó pontok koordinátái, illetve ha síkbeli is lenne...
- A háromszög ilyen!
- Minden egyéb felületet ilyen primitívekkel (lényegében: síklapokkal) közelítünk $\leftarrow$ *tesszelláció*

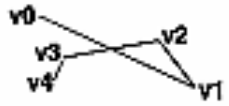
# Raszterizáció – primitívek



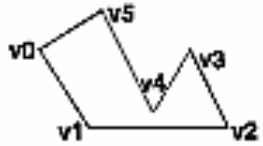
GL_POINTS



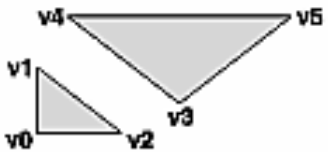
GL_LINES



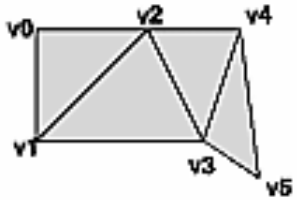
GL_LINE_STRIP



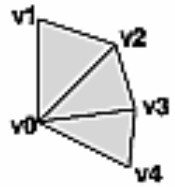
GL_LINE_LOOP



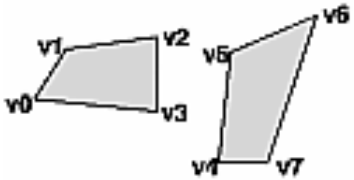
GL_TRIANGLES



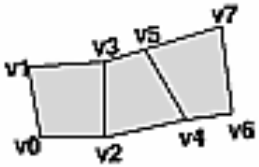
GL_TRIANGLE_STRIP



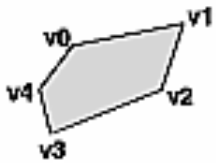
GL_TRIANGLE_FAN



GL_QUADS



GL_QUAD_STRIP



GL_POLYGON

# Raszterizáció – primitívek

A modern OpenGL-ben a GL_QUADS, GL_QUAD_STRIP, GL_POLYGON primitívek már nem használhatóak.

# Tartalom

# Takarási feladat

- Feladat: eldönteni, hogy a kép egyes részein milyen felületdarab látszik.

# Takarási feladat

- Feladat: eldönteni, hogy a kép egyes részein milyen felületdarab látszik.
- Objektum tér algoritmusok:
  - Logikai egységenként dolgozunk, nem függ a képernyő felbontásától.

# Takarási feladat

- Feladat: eldönteni, hogy a kép egyes részein milyen felületdarab látszik.
- Objektum tér algoritmusok:
  - Logikai egységenként dolgozunk, nem függ a képernyő felbontásától.
  - Rossz hír: nem fog menni.

# Takarási feladat

- Feladat: eldönteni, hogy a kép egyes részein milyen felületdarab látszik.
- Objektum tér algoritmusok:
  - Logikai egységenként dolgozunk, nem függ a képernyő felbontásától.
  - Rossz hír: nem fog menni.
- Képtér algoritmusok:
  - Pixelenként döntjük el, hogy mi látszik.

# Takarási feladat

- Feladat: eldönteni, hogy a kép egyes részein milyen felületdarab látszik.
- Objektum tér algoritmusok:
  - Logikai egységenként dolgozunk, nem függ a képernyő felbontásától.
  - Rossz hír: nem fog menni.
- Képtér algoritmusok:
  - Pixelenként döntjük el, hogy mi látszik.
  - Ilyen a sugárkövetés is.

# Tartalom

# Triviális hátlapeldobás, *Back-face culling*

- Feltételezés: az objektumaink „zártak”, azaz ha nem vagyunk benne az objektumban, akkor sehonnan sem láthatjuk a felületét belülről.

# Triviális hátlapeldobás, *Back-face culling*

- Feltételezés: az objektumaink „zártak”, azaz ha nem vagyunk benne az objektumban, akkor sehonnan sem láthatjuk a felületét belülről.
- Körüljárási irány: rögzítsük, hogy a poligonok csúcsait milyen sorrendben *kell* megadni:
  - óramutató járásával megegyező (*clockwise, CW*)
  - óramutató járásával ellentétes (*counter clockwise, CCW*)

# Triviális hátlapeldobás, *Back-face culling*

- Feltételezés: az objektumaink „zártak”, azaz ha nem vagyunk benne az objektumban, akkor sehonnan sem láthatjuk a felületét belülről.
- Körüljárási irány: rögzítsük, hogy a poligonok csúcsait milyen sorrendben *kell* megadni:
  - óramutató járásával megegyező (*clockwise, CW*)
  - óramutató járásával ellentétes (*counter clockwise, CCW*)
- Ha a transzformációk után a csúcsok sorrendje nem egyezik meg a megadással, akkor a lapot *hátulról* látjuk $\Rightarrow$ nem kell kirajzolni, *eldobható*.

# Tartalom

# Festő algoritmus

- Rajzoljuk ki hátulról előre haladva a poligonokat!

# Festő algoritmus

- Rajzoljuk ki hátulról előre haladva a poligonokat!
- Ami közelebb van, azt később rajzoljuk $\Rightarrow$ ami távolabb van, takarva lesz.

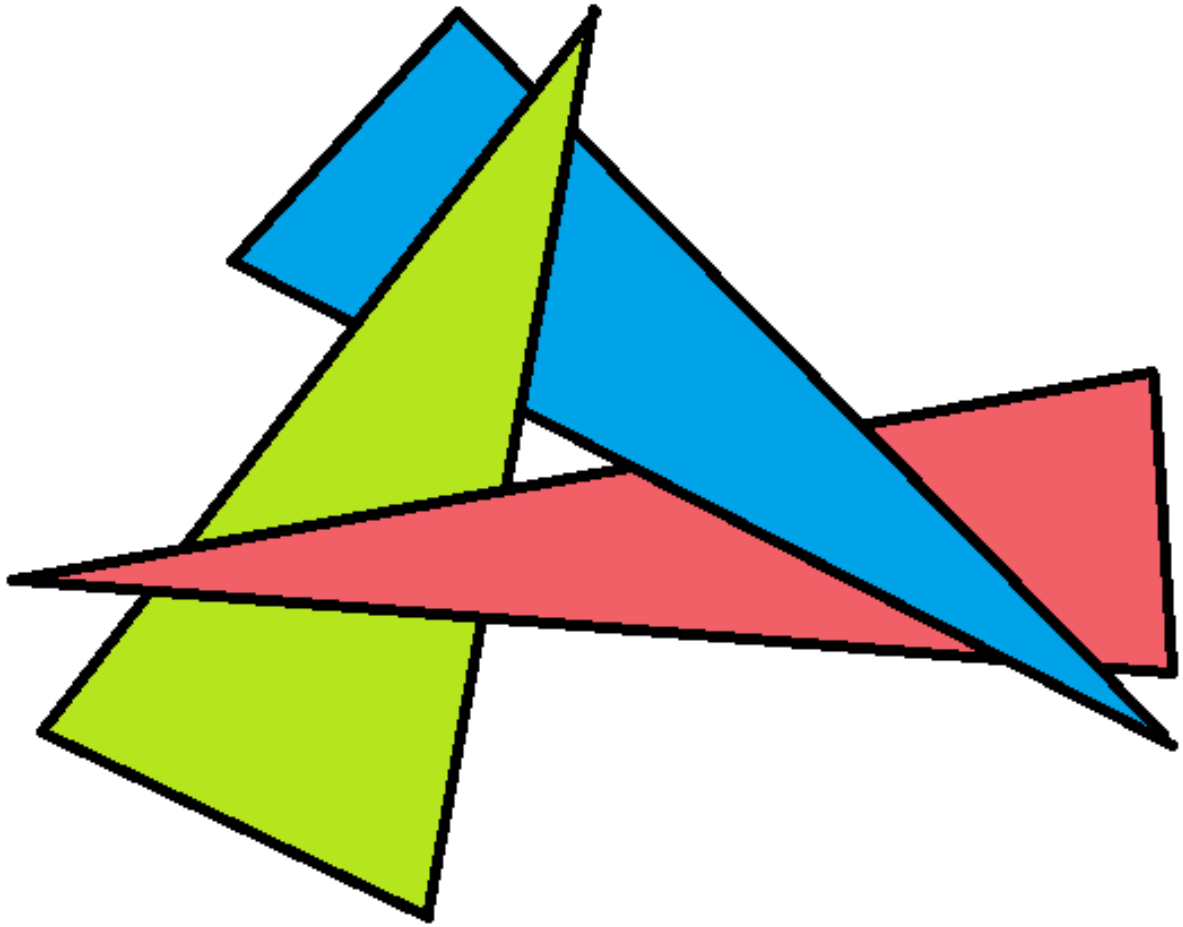
# Festő algoritmus

- Rajzoljuk ki hátulról előre haladva a poligonokat!
- Ami közelebb van, azt később rajzoljuk $\Rightarrow$ ami távolabb van, takarva lesz.
- Probléma: hogyan rakjuk sorrendbe a poligonokat?

# Festő algoritmus

- Rajzoljuk ki hátulról előre haladva a poligonokat!
- Ami közelebb van, azt később rajzoljuk $\Rightarrow$ ami távolabb van, takarva lesz.
- Probléma: hogyan rakjuk sorrendbe a poligonokat?
- Már néhány háromszögnél is előfordul olyan eset, amikor nem lehet egyértelmű sorrendet megadni.

# Festő algoritmus



# Tartalom

# Z-buffer algoritmus

- Képtérbeli algoritmus

# Z-buffer algoritmus

- Képtérbeli algoritmus
- Minden pixelre nyilvántartjuk, hogy ahhoz milyen mélységérték tartozott.

# Z-buffer algoritmus

- Képtérbeli algoritmus
- Minden pixelre nyilvántartjuk, hogy ahhoz milyen mélységérték tartozott.
- Ha megint újra erre a pixelre rajzolnánk (*Z-teszt*):
  - **Ha** az új Z érték mélyebben van, **akkor** ez a pont takarva van $\Rightarrow$ nem rajzolunk
  - **Ha** a régi Z érték mélyebben van, **akkor** az új pont kitakarja azt $\Rightarrow$ rajzolunk *és* eltároljuk az új Z értéket.

# Z-buffer

- Z-buffer vagy *depth buffer*: külön memóriaterület.

# Z-buffer

- Z-buffer vagy *depth buffer*: külön memóriaterület.
- Képernyő/ablak méretével megegyező méretű tömb.

# Z-buffer

- Z-buffer vagy *depth buffer*: külön memóriaterület.
- Képernyő/ablak méretével megegyező méretű tömb.
- Pontosság: a közeli és távoli vágósík közti távolságtól függ.

# Z-buffer

- Z-buffer vagy *depth buffer*: külön memóriaterület.
- Képernyő/ablak méretével megegyező méretű tömb.
- Pontosság: a közeli és távoli vágósík közti távolságtól függ.
- Minden pixelhez tartozik egy érték a bufferből.

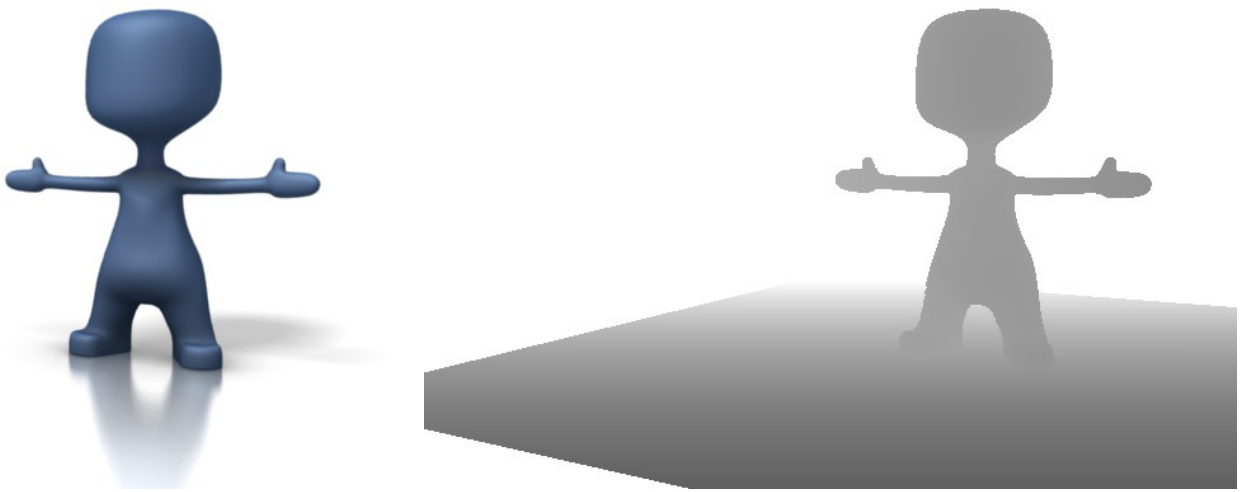
# Z-buffer

- Z-buffer vagy *depth buffer*: külön memóriaterület.
- Képernyő/ablak méretével megegyező méretű tömb.
- Pontosság: a közeli és távoli vágósík közti távolságtól függ.
- Minden pixelhez tartozik egy érték a bufferből.
- Ezzel kell összehasonlítani, és ide kell írni, ha a pixel *átment a Z-teszten*.

# Z-buffer

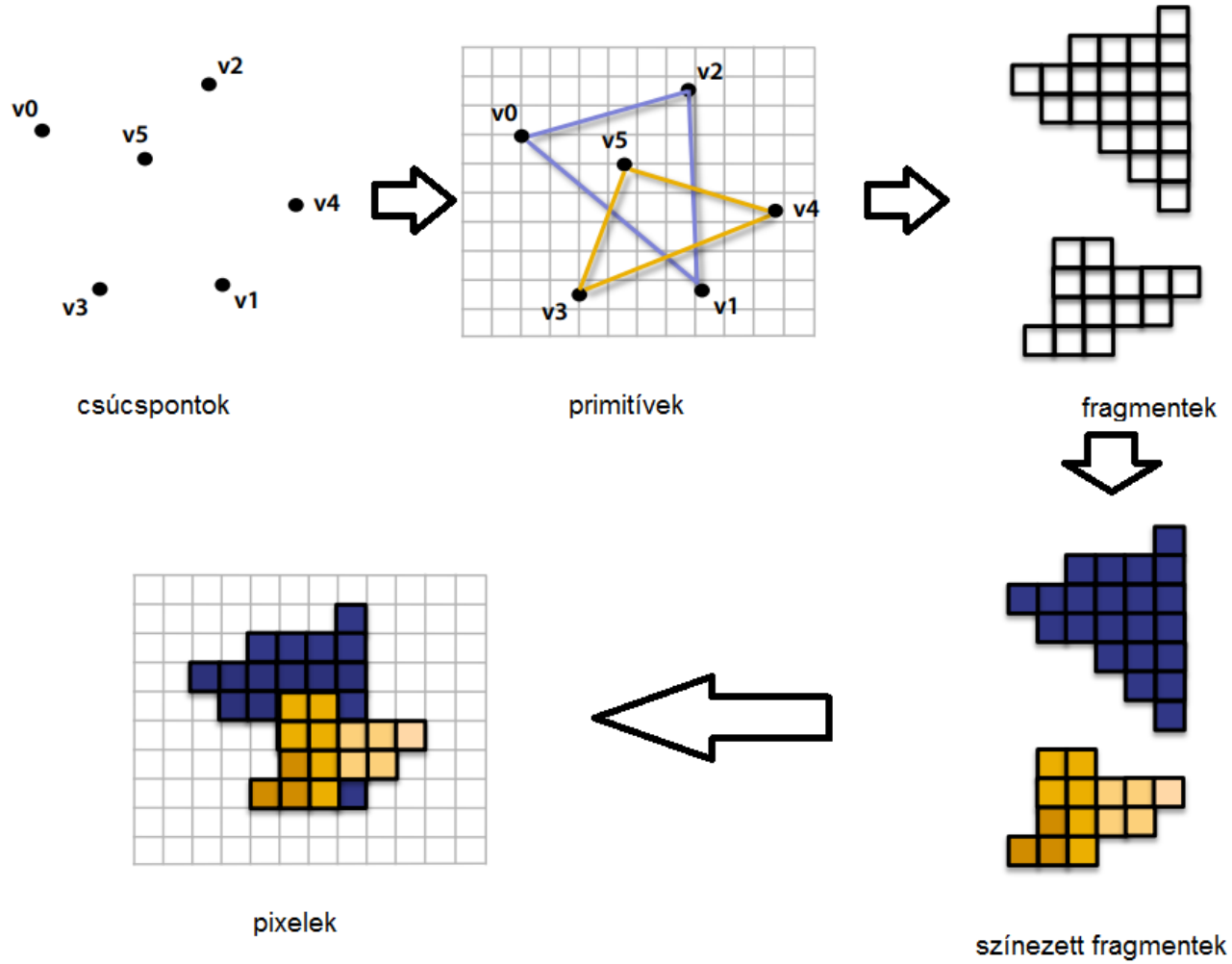
- Z-buffer vagy *depth buffer*: külön memóriaterület.
- Képernyő/ablak méretével megegyező méretű tömb.
- Pontosság: a közeli és távoli vágósík közti távolságtól függ.
- Minden pixelhez tartozik egy érték a bufferből.
- Ezzel kell összehasonlítani, és ide kell írni, ha a pixel *átment a Z-teszten*.
- Gyakorlatban:
  - 16-32 bites elemek
  - Hardveres gyorsítás
  - Pl. közeli vágósík: $t$, távoli: $1000t$, akkor a Z-buffer 98%-a a tartomány első 2%-át írja le.

# Z-buffer



by macouno, macouno.com

# GPU-s szerelőszalag



# Tartalom

# Lokális illumináció

- Ha már megvannak a primitívjeink pixelekre való leképezései, valahogyan számítsunk színeket

# Tartalom

# Saját színnel árnyalás

- Minden objektumhoz/primitívhez egy színt rendelünk, és kirajzoláskor ez lesz a pixelek értéke.

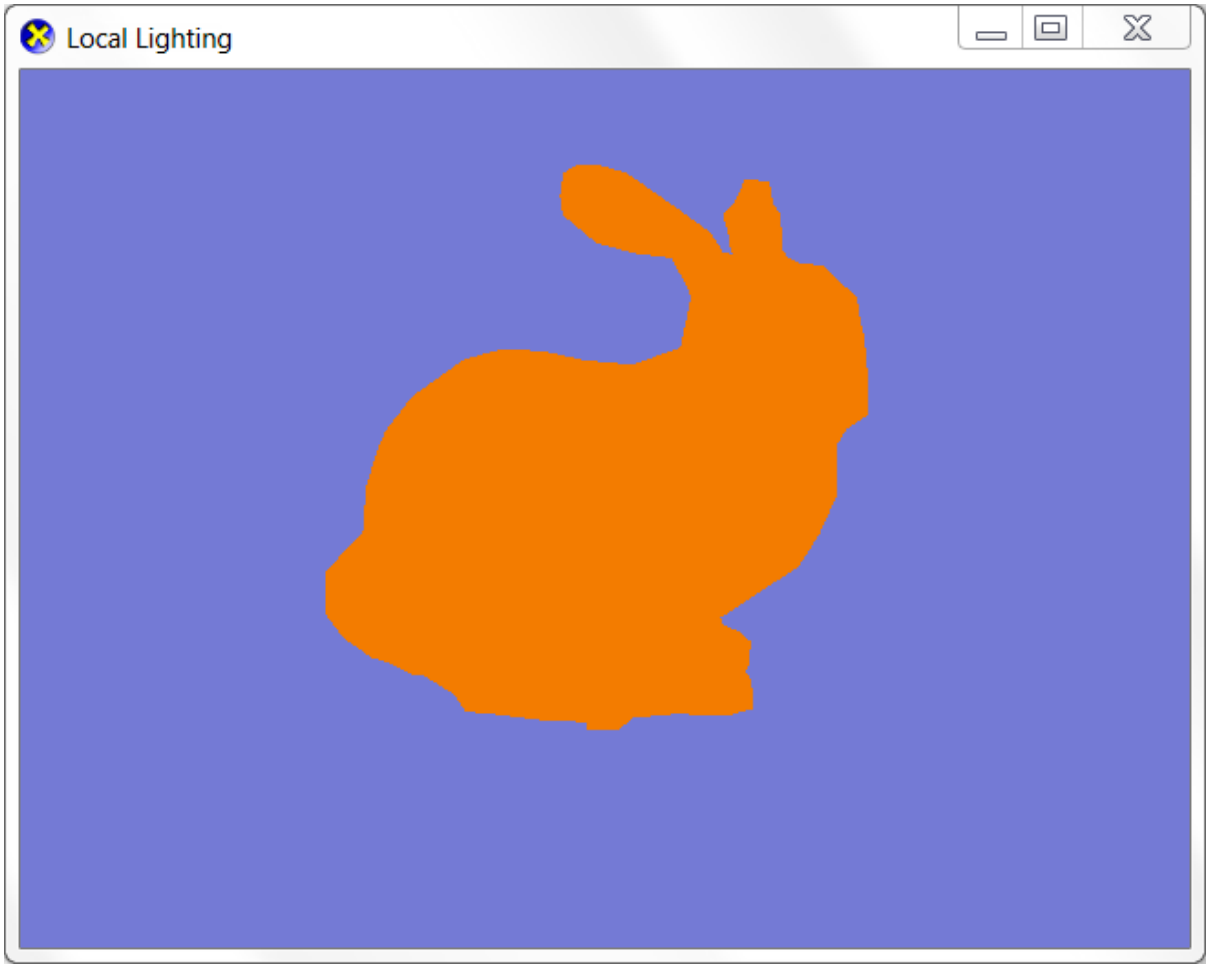
# Saját színnel árnyalás

- Minden objektumhoz/primitívhez egy színt rendelünk, és kirajzoláskor ez lesz a pixelek értéke.
- Leggyorsabb: az illumináció gyakorlatilag egyetlen értékadás.

# Saját színnel árnyalás

- Minden objektumhoz/primitívhez egy színt rendelünk, és kirajzoláskor ez lesz a pixelek értéke.
- Leggyorsabb: az illumináció gyakorlatilag egyetlen értékadás.
- Borzasztó: se nem valósághű, se nem szép.

# Saját színnel árnyalás



# Tartalom

# Konstans árnyalás, *Flat shading*

- A megvilágítást poligononként egyszer számítjuk ki, a szín homogén a lapon belül.

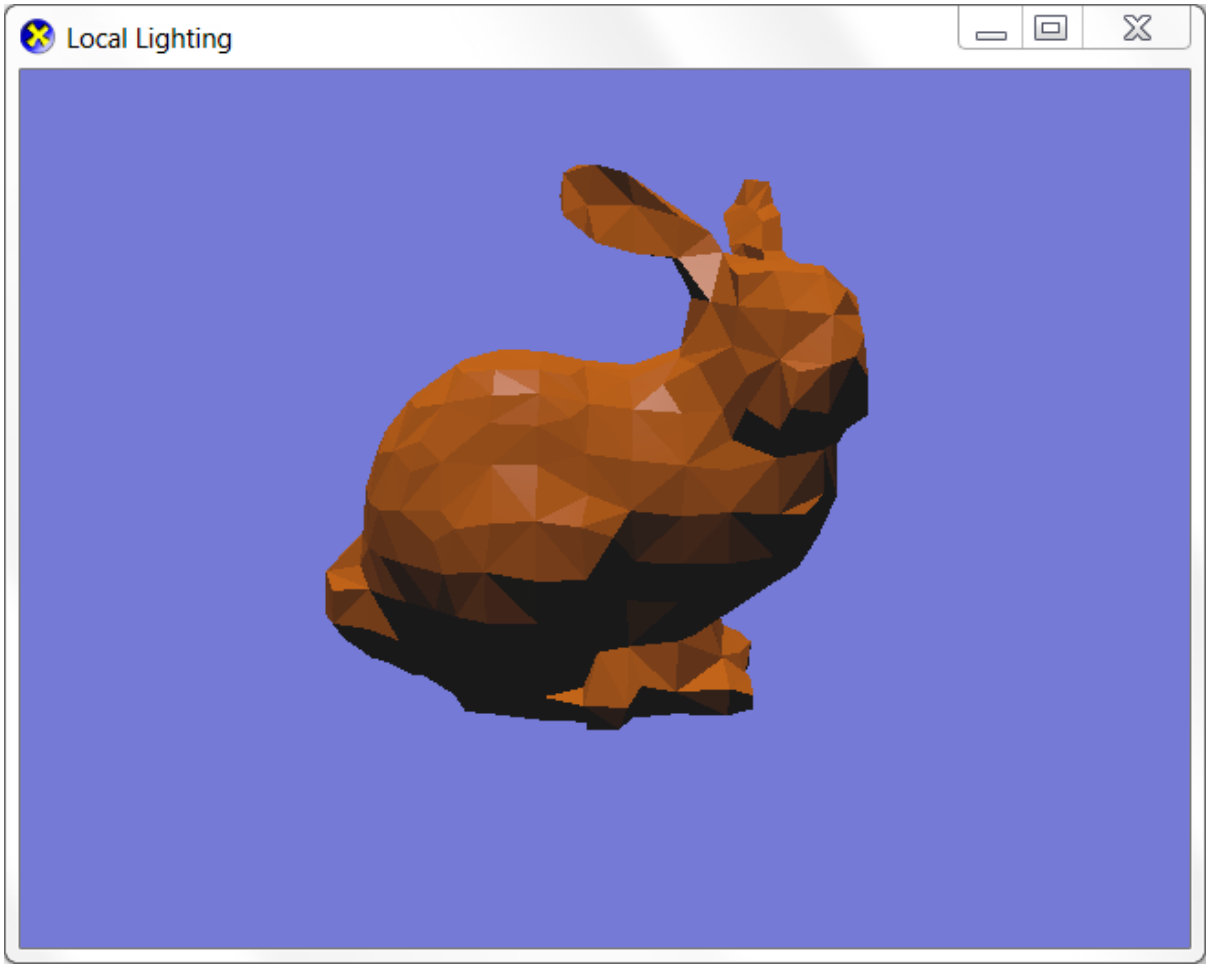
# Konstans árnyalás, *Flat shading*

- A megvilágítást poligononként egyszer számítjuk ki, a szín homogén a lapon belül.
- Gyors: a műveletek száma a poligonok számától függ, a pixelek számától független.

# Konstans árnyalás, *Flat shading*

- A megvilágítást poligononként egyszer számítjuk ki, a szín homogén a lapon belül.
- Gyors: a műveletek száma a poligonok számától függ, a pixelek számától független.
- Van hogy használható: íves részeket nem tartalmazó, diffúz, egyszínű objektumokra.

# Konstans árnyalás, *Flat shading*



# Tartalom

# Gouraud-árnyalás

- A megvilágítást csúcspontonként számítjuk ki, a lapon interpolációval számítjuk a színeket a három csúcspontban kapott értékből.

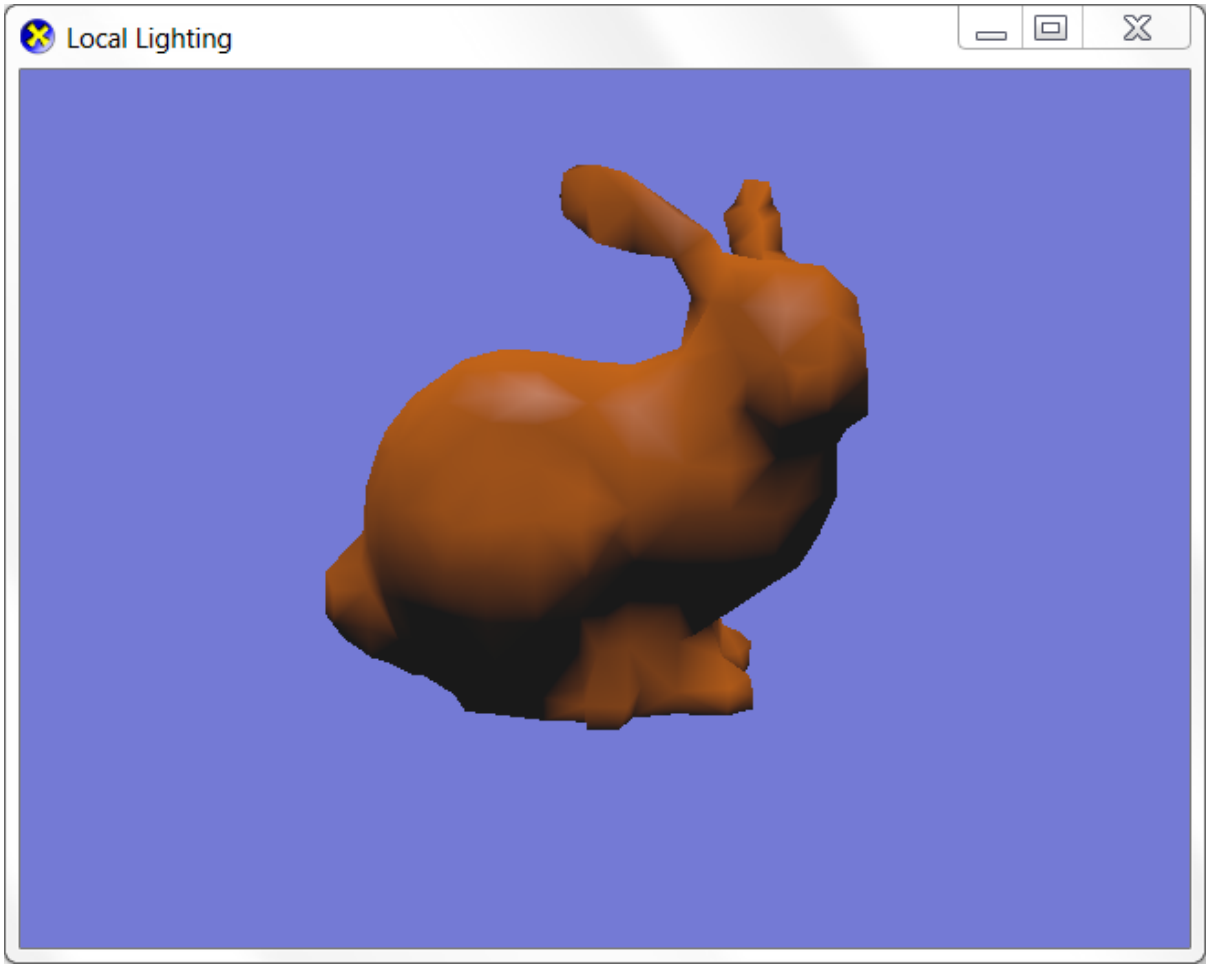
# Gouraud-árnyalás

- A megvilágítást csúcspontonként számítjuk ki, a lapon interpolációval számítjuk a színeket a három csúcspontban kapott értékből.
- Lassabb: $N$ db megvilágítás számítás + minden pixelre interpoláció.

# Gouraud-árnyalás

- A megvilágítást csúcspontonként számítjuk ki, a lapon interpolációval számítjuk a színeket a három csúcspontban kapott értékből.
- Lassabb: $N$ db megvilágítás számítás + minden pixelre interpoláció.
- Szebb: az árnyalás minősége nagyban függ a poligonok számától. De nagy lapokon nem tud megjelenni a csillanás.

# Gouraud-árnyalás



# Tartalom

# Phong-árnyalás

- Csak a normálvektorokat interpoláljuk, a megvilágítást minden pixelre kiszámítjuk.

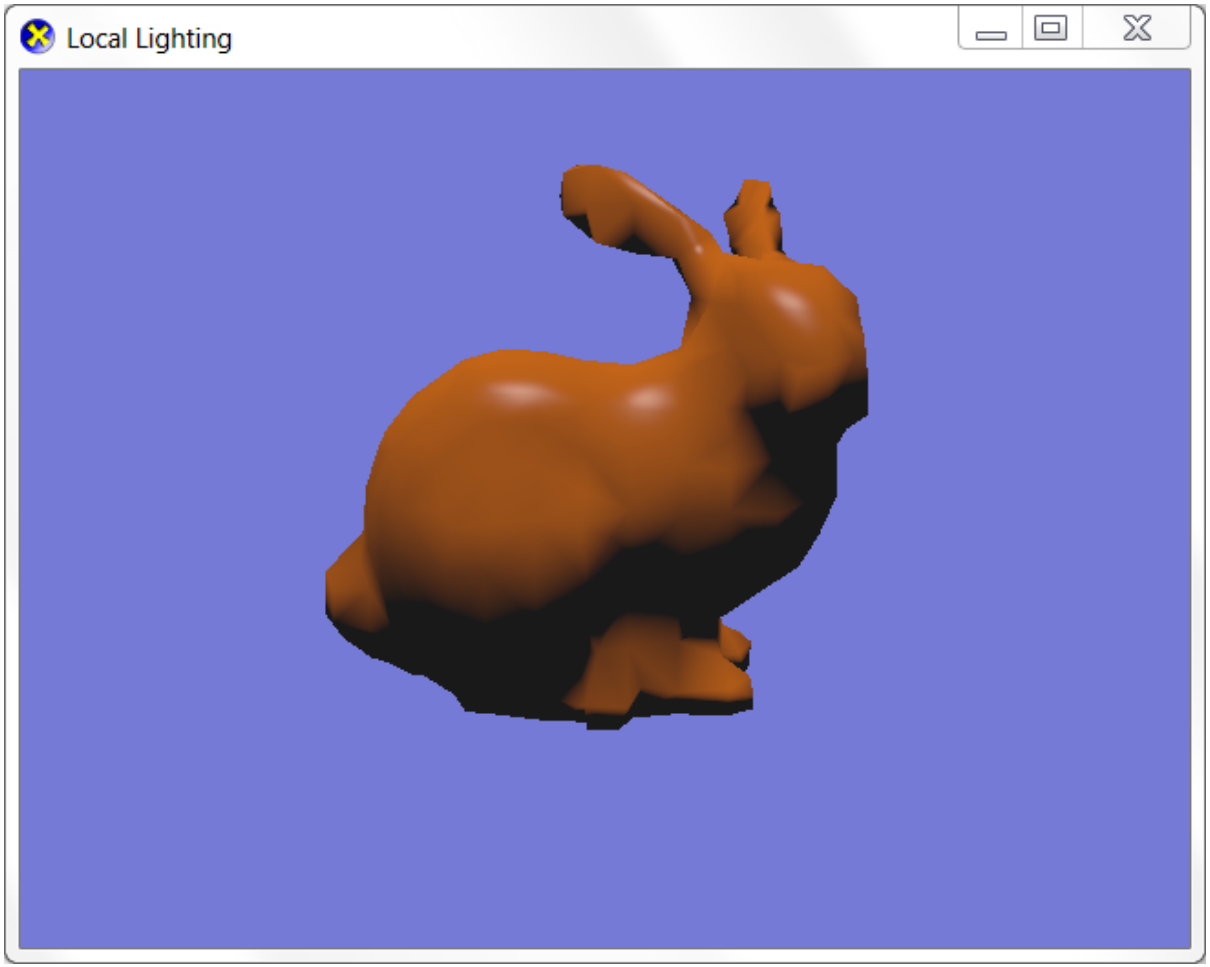
# Phong-árnyalás

- Csak a normálvektorokat interpoláljuk, a megvilágítást minden pixelre kiszámítjuk.
- Leglassabb: *pixelek száma* db megvilágítás számítás.

# Phong-árnyalás

- Csak a normálvektorokat interpoláljuk, a megvilágítást minden pixelre kiszámítjuk.
- Leglassabb: *pixelek száma* db megvilágítás számítás.
- Legszebb: az árnyalás minősége nem függ a poligonok számától. Csillanás akár a poligon közepén is meg tud jelenni.

# Phong-árnyalás



# Gouraud- vs Phong-árnyalás