Számítógépes Grafika BSc
8. gyakorlat

Kamera

Textúrák

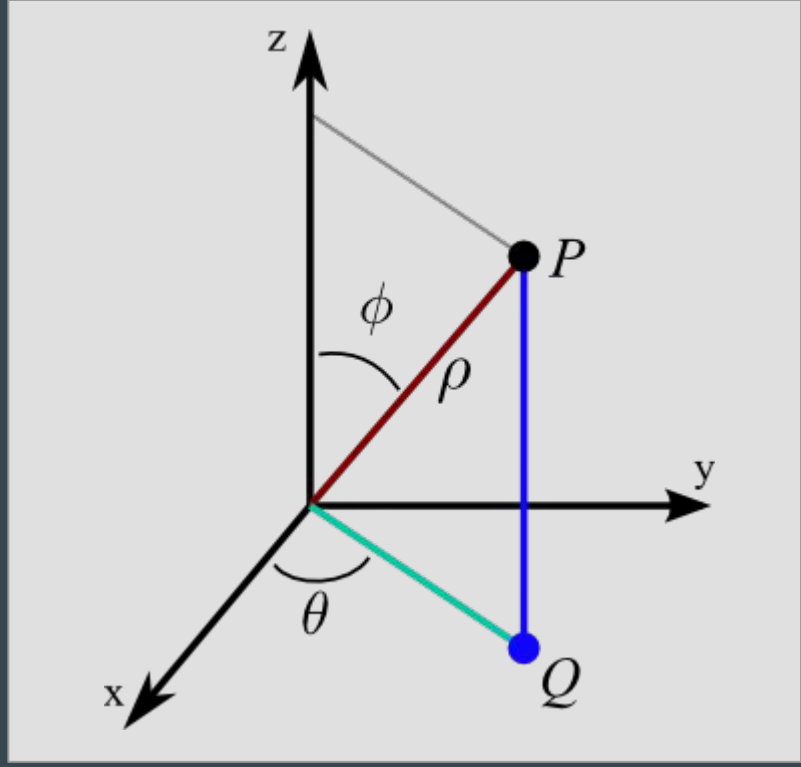
Projekt: http://cg.elte.hu/~bsc_cg/ujgyak/07/01_Textures.zip

Kamera megvalósítása

Két rész:

1. Kamera forgatása
2. Kamera mozgatása

Körülnézés



Gömbi koordináta rendszerben kijelölünk egy pontot, melyet fi és theta szög fog reprezentálni.
A kamera a gömb origójában van, a sugár tetszőleges, csak a nézeti irány számít.
A Fí és theta szöget az egérrel tudjuk változtatni.
Majd ezt átváltjuk Descartes koordináta rendszerbe.

Kamera megvalósítása, Kamera forgatása

Írjunk egy polárból descartesba váltó függvényt:

```
glm::vec3 CMyApp::toDesc(float fi, float theta) {
	return glm::vec3(sin(fi)*cos(theta), cos(fi), -sin(fi)*sin(theta));
}
```

Vegyük fel a header-be az osztálytagok közé a fi és theta változókat.

```
float m_fi = M_PI / 2.0;
float m_theta = M_PI / 2.0;
```

Majd inicializáljuk a jelenlegi szempozíciót, nézeti irányt és felfelé mutató vektort.

```
glm::vec3 m_eye = glm::vec3(0, 0, 10);
glm::vec3 m_at = m_eye + toDesc(m_fi, m_theta);
glm::vec3 m_up = glm::vec3(0, 1, 0);
```

Kamera megvalósítása, Kamera forgatása

Ne felejtsük átírni az update()-ben a nézeti tr. mátrixot gyártó függvény paramétereit!

```
m_matView = glm::lookAt(m_eye, m_at, m_up);
```

Majd az egérmozgatásnál frissítsük fi és theta értékét, illetve ezek alapján a nézeti irányt!

```
void CMyApp::MouseMove(SDL_MouseMotionEvent& mouse) {
	if (mouse.state & SDL_BUTTON_LMASK) {
		m_theta -= mouse.xrel*0.005;
		m_fi += mouse.yrel*0.005;
		m_fi = glm::clamp(m_fi, 0.001f, 3.13f);
		m_at = m_eye + toDesc(m_fi, m_theta);
	}
}
```

Kamera megvalósítása, Kamera mozgatása

A KeyboardDown()-ban és KeyboardUp()-ban csak mentsük el, hogy le van-e nyomva a WASD billentyűk közül valamelyik. Ehhez vegyünk fel a header-be 4 változót:

```
bool w = false, a = false, s = false, d = false;
```

Az eseménykezelőkben állítsuk be a változókat a jelentésüknek megfelelően.

```
void CMyApp::KeyboardDown(SDL_KeyboardEvent& key) {
	switch (key.keysym.sym) {
	case SDLK_w:
		w = true;
		break;
	case SDLK_s:
		s = true;
		break;
	case SDLK_a:
		a = true;
		break;
	case SDLK_d:
		d = true;
		break;
	}
}
```

```
void CMyApp::KeyboardUp(SDL_KeyboardEvent& key) {
	switch (key.keysym.sym) {
	case SDLK_w:
		w = false;
		break;
	case SDLK_s:
		s = false;
		break;
	case SDLK_a:
		a = false;
		break;
	case SDLK_d:
		d = false;
		break;
	}
}
```

Kamera megvalósítása, Kamera mozgatása

Az előbbieket felhasználva mozgathatjuk a kamerát az Update()-ben. Számoljuk ki az előre mutató irányt. Ezzel eltoljuk a kamerapozíciót ha kell, és frissítjük, hogy hova nézünk.

```
glm::vec3 forward = glm::normalize(m_at - m_eye);
if (w) {
	m_eye += 0.1f * forward;
	m_at = m_eye + toDesc(m_fi, m_theta);
}
if (s) {
	m_eye -= 0.1f * forward;
	m_at = m_eye + toDesc(m_fi, m_theta);
}
```

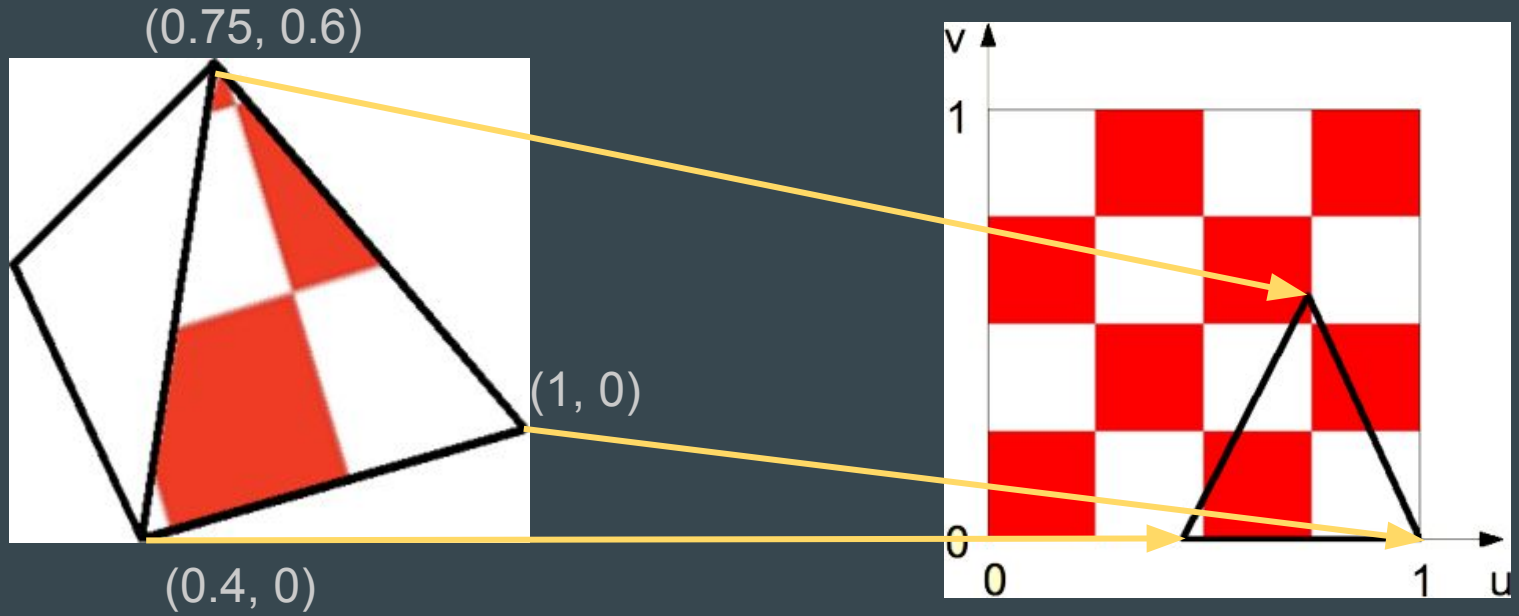
Kamera megvalósítása, Kamera mozgatása

A jobbra és balra mozgatáshoz keresztszorozzuk az előre és felfelé mutató irányt.
Ez merőleges lesz az előbbi kettőre, és sorrendtől függően jobbra vagy balra mutat.

```
glm::vec3 left = glm::normalize(glm::cross(m_up, forward));
if (a) {
	m_eye += 0.1f * left;
	m_at = m_eye + toDesc(m_fi, m_theta);
}
if (d) {
	m_eye -= 0.1f * left;
	m_at = m_eye + toDesc(m_fi, m_theta);
}
```

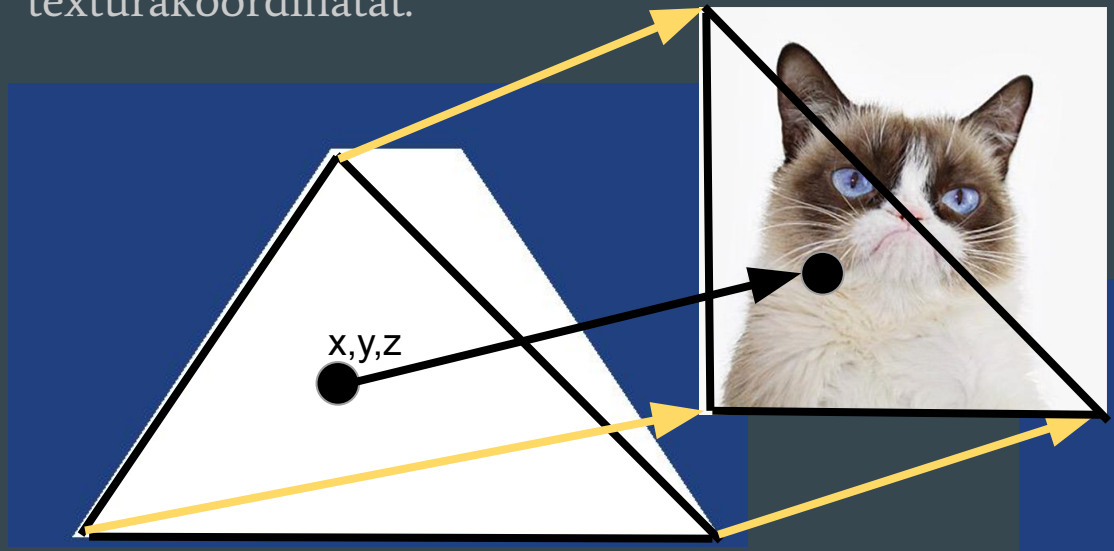
Vertex - textúrakoordináta attribútum

A vertexeinkben a szín helyett ezentúl egy textúra-koordinátát tárolunk. Ez egy 2D-s vektor amely megadja, hogy a textúra melyik pontja tartozik a vertexhez.



Baricentrikus koordináták

Ekkor hasonlóan, mint a színnél tudjuk interpolálni a háromszög fragmentjeire textúrakoordinátát.

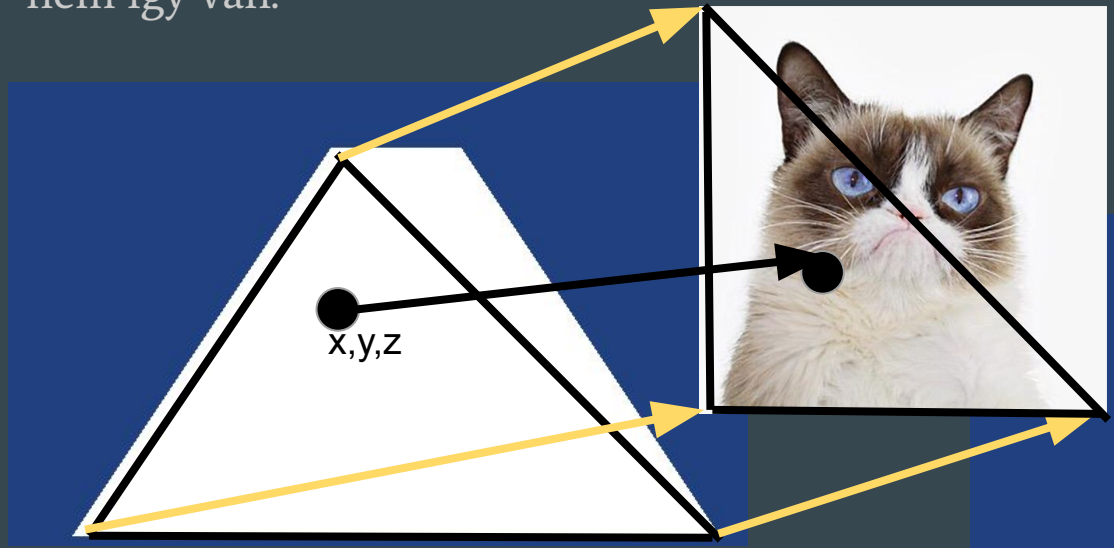


Ha síkban képzeljük el az alakzatot, akkor láthatjuk, hogy az egyik háromszög sokkal nagyobb területű, mint a másik...

Hoppá! Ha csak simán interpoláljuk a textúra koordinátákat a háromszög fragmentjének baricentrikus koordinátái alapján, akkor a perspektíva miatt érdekes dolgot kapunk

Baricentrikus koordináták

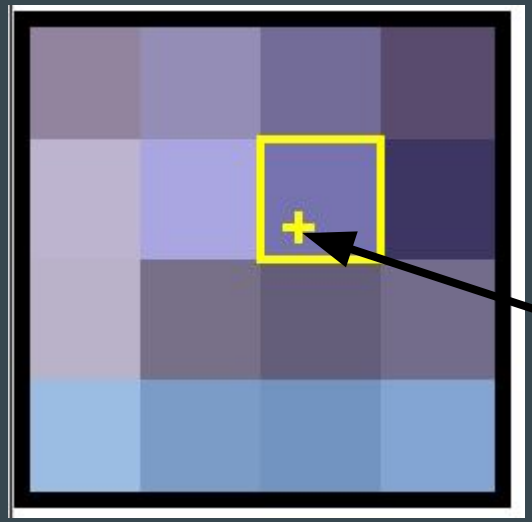
Színnél még nem tűnik fel, de textúránál már igen, ha affin interpolációt végzünk. (azaz úgy tekintjük, mint ha a vertexeken csak affin transzformációt végeztünk volna). Ami nem így van.



Perspektíva helyes interpoláció

Szerencsére a vertex és fragment shader között alapból így történik az interpoláció.

Textúra szűrés



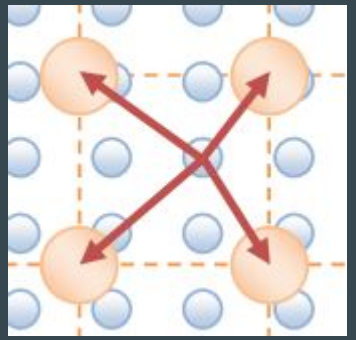
Texel: a textúra egy pixele.
A textúra koordináták interpolálásából nem feltétlenül egész szám jön ki (sőt meg is lepődnénk, ha igen), nekünk viszont valamelyik konkrét texel színére van szükségünk, ami a fragment színe lesz.

Egy interpolálással kapott koordináta.
Triviális megközelítésnek tűnik, hogy a sárga négyzettel körbevett texelt válasszuk a színnek.

Textúra szűrés

Az előbbi módszert a legközelebbi szomszéd módszernek nevezzük (GL_NEAREST)
A textúra koordináta és a texelek középpontjai között Manhattan távolsággal veszi a legközelebbit.

GL_LINEAR: bilineáris interpoláció (szűrés), a négy szomszédos texelből súlyozott (távolság) átlaggal számítja ki a színt



Egyik szemcsés, másik homályos… Van jobb megoldás?
Van! Trilineáris szűrés (következő dia)

Trilineáris szűrés

Ez sem a legjobb szűrés, de nekünk ennyi elég lesz

MIPMAP-ek segítségével történik. Legenerálunk előre az eredeti textúránknak megfelelő, egyre kisebb textúrákat (pl Box filterrel). Bilineárisan interpolálunk két textúrát (valahogy meghatározzuk a megfelelő LODot), majd ezek között lineárisan interpolálunk. Ha LOD0 fölé (vagy LODmax alá) megyünk, akkor csak bilineáris szűrésről beszélünk, hisz csak egy textúrából veszünk mintát.

Adott level mintavételezése

MIP levelek közötti interpoláció

GL_{LINEAR, NEAREST}_MIPMAP_{LINEAR, NEAREST}

Textúra létrehozása

Ugyanúgy textúra erőforrás-azonosítót kell generálnunk, mint a VAO, VBO esetében. Ez egy szám lesz.

Legeneráljuk a MIPMAPeket, ha esetleg segítségükkel szeretnénk szűrni.

```
glGenTextures(1, &tmpID); // új textúra

glBindTexture(GL_TEXTURE_2D, tmpID); // aktiváljuk

glTexImage2D(GL_TEXTURE_2D,...); // feltöltjük az adatokat

glGenerateMipmap(GL_TEXTURE_2D); // MIPMAP szintek generálása

glTexParameteri(GL_TEXTURE_2D,...); // mintavételezési beállítások
```

Textúra mintavételezési beállításai

- `GL_TEXTURE_MAG_FILTER` – nagyítás
- `GL_TEXTURE_MIN_FILTER` – kicsinyítés
- `GL_TEXTURE_WRAP_S` – mi történjen, ha vízszintesen kilépünk a [0,1] tartományból
- `GL_TEXTURE_WRAP_T` – ugyanez függőlegesen
- ... és még sok más

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_LINEAR);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

Textúrák használata OGL-ben

Először aktiváljuk a 0-ás textúra mintavételező egységet (az OGL minimum 80-at (?) biztosít számunkra), az ez után következő mintavételezőre vonatkozó utasítások a 0-ás textúra mintavételezőn fognak végrehajtódni.

```
glActiveTexture(GL_TEXTURE0);
```

Hozzárendeljük a 0-ás mintavételezőhöz a kívánt textúrát (`textúra ID`).

```
glBindTexture(GL_TEXTURE_2D, textúra ID);
```

A fragment shadernek pedig átadjuk uniform paraméterben, hogy a 0-ás mintavételezőt használja az olvasáshoz.

```
glUniform1i(m_loc_texture, 0);
```

Sampler2D

Igaz, hogy a fragment shadernek egy számot adunk át, hogy mely mintavételezővel olvassa a textúrát, de ez magában a shaderben `Sampler2D` típust takar.

A fragment shaderben a `texture` függvénnyel vehetünk mintát a textúrából.

```
texture(sampler, textúra koordináta)
```

Sampler objektum

Azt, hogy a textúrából hogyan olvasunk, egy sampler objektummal felülírhatjuk. Ehhez is egy szokásos erőforrás-azonosító fog tartozni.

```
glGenSamplers(1, &samp);
```

Hasonló a beállítása, mint a textúrának:

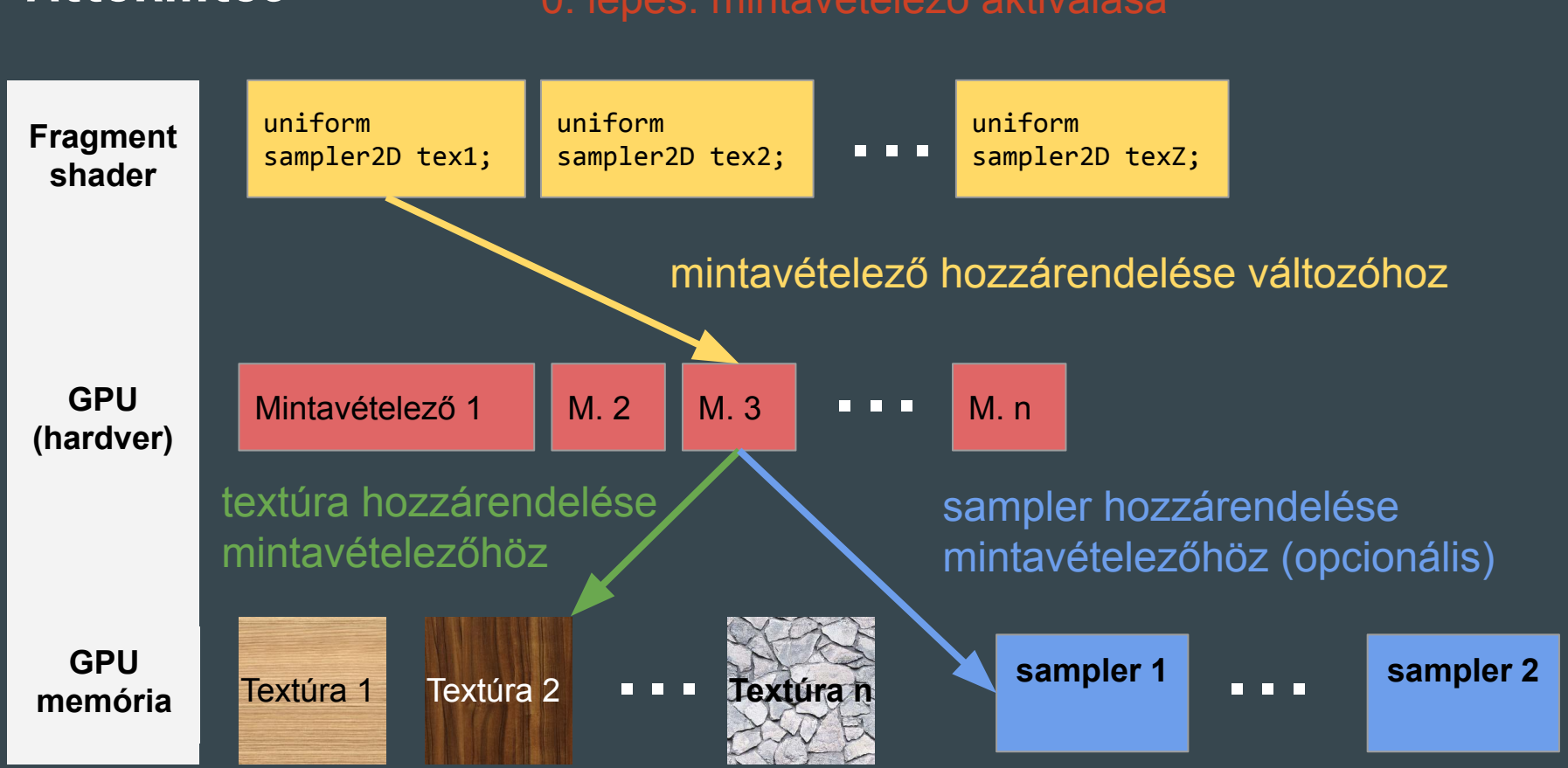
```
glSamplerParameteri(samp, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glSamplerParameteri(samp, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
```

Használat (0-ás mintavételező egységhez csatolás):

```
glBindSampler(0, samp);
```

A sampler több mintavételezőhöz is csatolható egyszerre. A sampler **nem** a textúrához kapcsolódik! Ha egy mintavételező egységnél újra a textúra saját beállításait akarjuk használni, akkor csatoljuk a 0-ás sampler-t.

Áttekintés


0. lépés: mintavételező aktiválása
Fragment shader
uniform sampler2D tex1;
uniform sampler2D tex2;
uniform sampler2D texZ;
mintavételező hozzárendelése változóhoz
GPU (hardver)
Mintavételező 1
M. 2
M. 3
M. n
textúra hozzárendelése mintavételezőhöz
sampler hozzárendelése mintavételezőhöz (opcionális)
GPU memória
Textúra 1
Textúra 2
Textúra n
sampler 1
sampler 2