14. fejezet

OpenCL textúra használat

Grafikus Processzorok Tudományos Célú Programozása

Berényi Dániel – Nagy-Egri Máté Ferenc

Textúrák

A textúrák 1, 2, vagy 3D-s tömbök kifejezetten szín információk tárolására

Főbb különbségek a bufferekhez képest:

- Célra szabott egyedi formátumok
  (1-4 csatorna, 8-32 bit szélesség, vagy float, azonban double nincs!)
- A memóriavezérlő után már más egységek kezelik, plussz műveleteket tudnak rajtuk végezni (pl.: lineáris interpoláció)
- Az adott dimenzióra felkészített cache logika
  (nagyon gyors elérni a szomszédos értékeket)

Textúrák

Az előzőekből következik, hogy a textúrák alapvetően képfeldolgozási feladatokra hasznosak.

Azonban előfordulhat, hogy valamilyen számítást elég float-ban elvégeznünk, ebben az esetben hasznos lehet őket használni

A textúra szűrési műveletek „ingyen” vannak abban az értelemben, hogy a kártya nominális teljesítményébe nincsenek bele értve.

Textúrák OpenCL-ben

OpenCL-ben a textúrát egy formátum és egy leíró párosával hozhatjuk létre:

```
cl_image_format format = { CL_RGBA, CL_FLOAT };
cl_image_desc desc = {};
desc.image_type = CL_MEM_OBJECT_IMAGE2D;
desc.image_width =  W; //x
desc.image_height = H; //y
desc.image_depth =  0; //z

cl_mem img = clCreateImage(context,
CL_MEM_READ_ONLY  | CL_MEM_COPY_HOST_PTR | CL_MEM_HOST_NO_ACCESS,
&format, &desc, image_data.data(), &status);
```

Textúrák OpenCL-ben

OpenCL-ben a textúrát egy formátum és egy leíró párosával hozhatjuk létre:

```
cl_image_format format = { CL_RGBA, CL_FLOAT };
cl_image_desc desc = {};
desc.image_type = CL_MEM_OBJECT_IMAGE2D;
desc.image_width =  W; //x
desc.image_height = H; //y
desc.image_depth =  0; //z

cl_mem img = clCreateImage(context,
CL_MEM_READ_ONLY  | CL_MEM_COPY_HOST_PTR | CL_MEM_HOST_NO_ACCESS,
&format, &desc, image_data.data(), &status);
```

Csatornák száma (most 4) és a csatorna reprezentációja (mot float)

Textúrák OpenCL-ben

WIGNER GPU Lab

OpenCL-ben a textúrát egy formátum és egy leíró párosával hozhatjuk létre:

```
cl_image_format format = { CL_RGBA, CL_FLOAT };
cl_image_desc desc = {};
desc.image_type = CL_MEM_OBJECT_IMAGE2D;
desc.image_width =  W; //x
desc.image_height = H; //y
desc.image_depth =  0; //z

cl_mem img = clCreateImage(context,
CL_MEM_READ_ONLY  | CL_MEM_COPY_HOST_PTR | CL_MEM_HOST_NO_ACCESS,
&format, &desc, image_data.data(), &status);
```

Kép dimenzióinak megadása

Textúrák OpenCL-ben

OpenCL-ben a textúrát egy formátum és egy leíró párosával hozhatjuk létre:

```
cl_image_format format = { CL_RGBA, CL_FLOAT };
cl_image_desc desc = {};
desc.image_type = CL_MEM_OBJECT_IMAGE2D;
desc.image_width =  W; //x
desc.image_height = H; //y
desc.image_depth =  0; //z

cl_mem img = clCreateImage(context,
CL_MEM_READ_ONLY  | CL_MEM_COPY_HOST_PTR | CL_MEM_HOST_NO_ACCESS,
&format, &desc, image_data.data(), &status);
```

Kép méreteinek megadása
A nem használt dimenziókba 0-t kell írni!

Textúrák OpenCL-ben

OpenCL-ben a textúrát egy formátum és egy leíró párosával hozhatjuk létre:

```
cl_image_format format = { CL_RGBA, CL_FLOAT };
cl_image_desc desc = {};
desc.image_type = CL_MEM_OBJECT_IMAGE2D;
desc.image_width =  W; //x
desc.image_height = H; //y
desc.image_depth =  0; //z
```

```
cl_mem img = clCreateImage(context,
CL_MEM_READ_ONLY  | CL_MEM_COPY_HOST_PTR | CL_MEM_HOST_NO_ACCESS,
&format, &desc, image_data.data(), &status);
```

Textúrák OpenCL-ben

OpenCL-ben a textúrát egy formátum és egy leíró párosával hozhatjuk létre:

```
cl_image_format format = { CL_RGBA, CL_FLOAT };
cl_image_desc desc = {};
desc.image_type = CL_MEM_OBJECT_IMAGE2D;
desc.image_width =  W; //x
desc.image_height = H; //y
desc.image_depth =  0; //z
```

```
cl_mem img = clCreateImage(context,
CL_MEM_READ_ONLY  | CL_MEM_COPY_HOST_PTR | CL_MEM_HOST_NO_ACCESS,
&format, &desc, image_data.data(), &status);
```

Textúrák OpenCL-ben

OpenCL-ben a textúrát egy formátum és egy leíró párosával hozhatjuk létre:

```
cl_image_format format = { CL_RGBA, CL_FLOAT };
cl_image_desc desc = {};
desc.image_type = CL_MEM_OBJECT_IMAGE2D;
desc.image_width =  W; //x
desc.image_height = H; //y
desc.image_depth =  0; //z

cl_mem img = clCreateImage(context,
CL_MEM_READ_ONLY  | CL_MEM_COPY_HOST_PTR | CL_MEM_HOST_NO_ACCESS,
&format, &desc, image_data.data(), &status);
```

Ez a pointer az adatokra, aminek most esetünkben W*H*4 db float-ra kell mutatnia

Textúrák OpenCL-ben

Az OpenCL kernelnek ugyanúgy kell átadni a textúrákat, mint a bufferesket:

```
cl_mem img;
cl_int status = clSetKernelArg(kernel, 0, sizeof(img), &img);
```

A kernel kódban pedig meg kell jelölni, hogy hány dimenziós képet várunk, és azt is, hogy írjuk vagy olvassuk (egyszerre a kettőt nem lehet!):

```
__kernel void sobel(read_only  image2d_t src,
                    write_only image2d_t dst){ ... }
```

Textúrák OpenCL-ben

Fel-, letölteni vagy másolni hasonló parancsokkal lehet, mint a buffereket, pl.:

```
size_t origin[3] = {0, 0, 0};
size_t dims[3] = {W, H, 1};

status = clEnqueueReadImage(queue, img, false, origin, dims,
0, 0, image_data.data(), 0, nullptr, nullptr);
```

Textúrák OpenCL-ben

WIGNER GPU Lab

Fel-, letölteni vagy másolni hasonló parancsokkal lehet, mint a buffereket, pl.:

A képnek egy rész „tégláját” is hivatkozhatjuk, a másolás kezdőpontjának, és a méretének megadásával. A nem használt dimenzióban a méret 1 kell legyen!

```
size_t origin[3] = {0, 0, 0};
size_t dims[3] = {W, H, 1};
```

Mi most az egész képet akarjuk másolni.

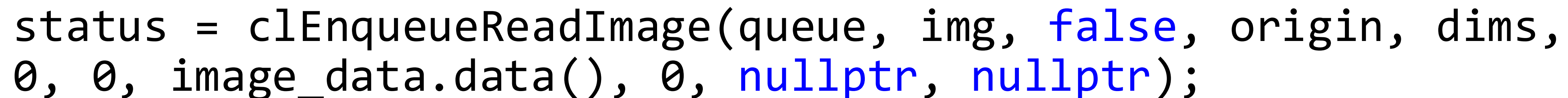
```
status = clEnqueueReadImage(queue, img, false, origin, dims,
0, 0, image_data.data(), 0, nullptr, nullptr);
```

Textúrák OpenCL-ben

Fel-, letölteni vagy másolni hasonló parancsokkal lehet, mint a buffereket, pl.:

```
size_t origin[3] = {0, 0, 0};
size_t dims[3] = {W, H, 1};
```

Blokkoló hívás, vagy sem (csak akkor tér-e vissza, ha befejezőtöbb a művelet, vagy pedig azonnal visszatér, amint bekerült a queue-be)



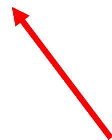
```
status = clEnqueueReadImage(queue, img, false, origin, dims,
0, 0, image_data.data(), 0, nullptr, nullptr);
```

Textúrák OpenCL-ben

Fel-, letölteni vagy másolni hasonló parancsokkal lehet, mint a buffereket, pl.:

```
size_t origin[3] = {0, 0, 0};
size_t dims[3] = {W, H, 1};
```

```
status = clEnqueueReadImage(queue, img, false, origin, dims,
0, 0, image_data.data(), 0, nullptr, nullptr);
```



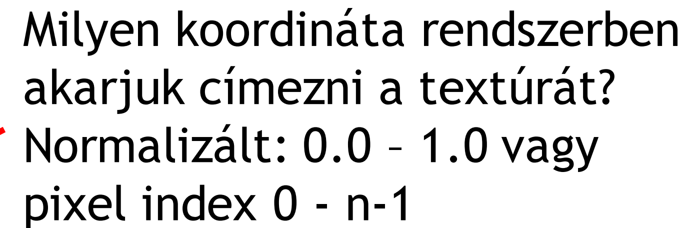
Pointer, ahova kérjük a kártyáról visszamásolt adatokat

Textúrák OpenCL-ben

Ahhoz, hogy egy textúrát olvasni tudjuk, meg kell adni, hogy milyen módon akarjuk mintavételezni (sampling), ez egy sampler objektum írja le.

Ezt mind a gazda oldalon, mind a kernelben definiálhatjuk, itt most az utóbbira van példa:

Milyen koordináta rendszerben akarjuk címezni a textúrát?
Normalizált: 0.0 – 1.0 vagy
pixel index 0 - n-1



```
__constant sampler_t sampler =
      CLK_NORMALIZED_COORDS_FALSE
    | CLK_ADDRESS_CLAMP_TO_EDGE
    | CLK_FILTER_NEAREST;
```

Textúrák OpenCL-ben

Ahhoz, hogy egy textúrát olvasni tudjuk, meg kell adni, hogy milyen módon akarjuk mintavételezni (sampling), ez egy sampler objektum írja le.

Ezt mind a gazda oldalon, mind a kernelben definiálhatjuk, itt most az utóbbira van példa:

```
__constant sampler_t sampler =
        CLK_NORMALIZED_COORDS_FALSE
      | CLK_ADDRESS_CLAMP_TO_EDGE
      | CLK_FILTER_NEAREST;
```

Mi történjén, ha túl indexeljük a széleken?
Repeat: periodikus határfeltétel
Clamp to edge: levágja a legközelebbi határra
Clamp: feketét ad vissza
None: undefined

Textúrák OpenCL-ben

Ahhoz, hogy egy textúrát olvasni tudjuk, meg kell adni, hogy milyen módon akarjuk mintavételezni (sampling), ez egy sampler objektum írja le.

Ezt mind a gazda oldalon, mind a kernelben definiálhatjuk, itt most az utóbbira van példa:

```
__constant sampler_t sampler =
        CLK_NORMALIZED_COORDS_FALSE
      | CLK_ADDRESS_CLAMP_TO_EDGE
      | CLK_FILTER_NEAREST;
```

A textúra szűrésének módja: Legközelebbi szomszéd, vagy lineáris interpoláció

Textúrák OpenCL-ben

WIGNER GPU Lab

Ezek után a read_image beépített függvénnyel lehet egy adott textúrából egy kiválasztott sampler-el beolvasni egy pixelt:

```
float4 pixel = read_imagef(src, sampler, (int2)(x, y));
```

Íráskor nincs szűrés, de ott is beépített függvény kell hívnunk:

```
write_imagef(dst, (int2)(x, y), pixel);
```

Példa: élkiemelés

WIGNER GPU Lab

A példakódban egy élkiemelési szűrést hajtunk végre a képen, aminek a neve Sobel-filter, ez két darab (egyik függőleges, a másik vízszintes) 3x3-as kernel (a képfeldolgoázsi értelemben☺), amivel konvolválni kell (szintén a képfeldolgozási értelemben!):

$$pixel_k(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} image(x+i, y+i) \cdot filter_k(i+1, j+1)$$

ahol most k=W vagy H.

Példa: élkiemelés

A két Sobel filter eredményének a négyzetösszegével kell korrigálni az eredeti képet, hogy a gradienst közelítsük:

$$gradient(x,y) = \sqrt{pixel_W(x,y)^2 + pixel_H(x,y)^2}$$

A példakódban a gradienst mind a 4 csatornában nézzük, ezért a négyzetre emelést a két 4-es vektor skaláris szorzatával számoljuk: `dot(pixel, pixel)`, ahol pixel egy `float4`!

Példa: élkiemelés

Wigner GPU Lab

Végül a gradiens értékét hozzá keverjük a képhez, hogy ne csak az éleket lássuk, hanem az eredeti képből is maradjon valami:

$$final_{pixel(x,y)} = pixel \cdot (\alpha + (1 - \alpha) \cdot gradient)$$

Példa: élkiemelés

Eredmény (az átlátszósági csatornát is szűrtük, így egy egész vicces sematikus ábrát nyertünk):