



Information Coding / Computer Graphics, ISY, LiTH

Compute shaders

Framtiden för GPU computing eller sen efterapning
av Direct Compute?

Tidigare rent Microsoft-koncept, Direct Compute

Senare även i OpenGL, ny shadertyp från OpenGL 4.3



Starkt alternativ

Varför använda det i stället för CUDA eller OpenCL?

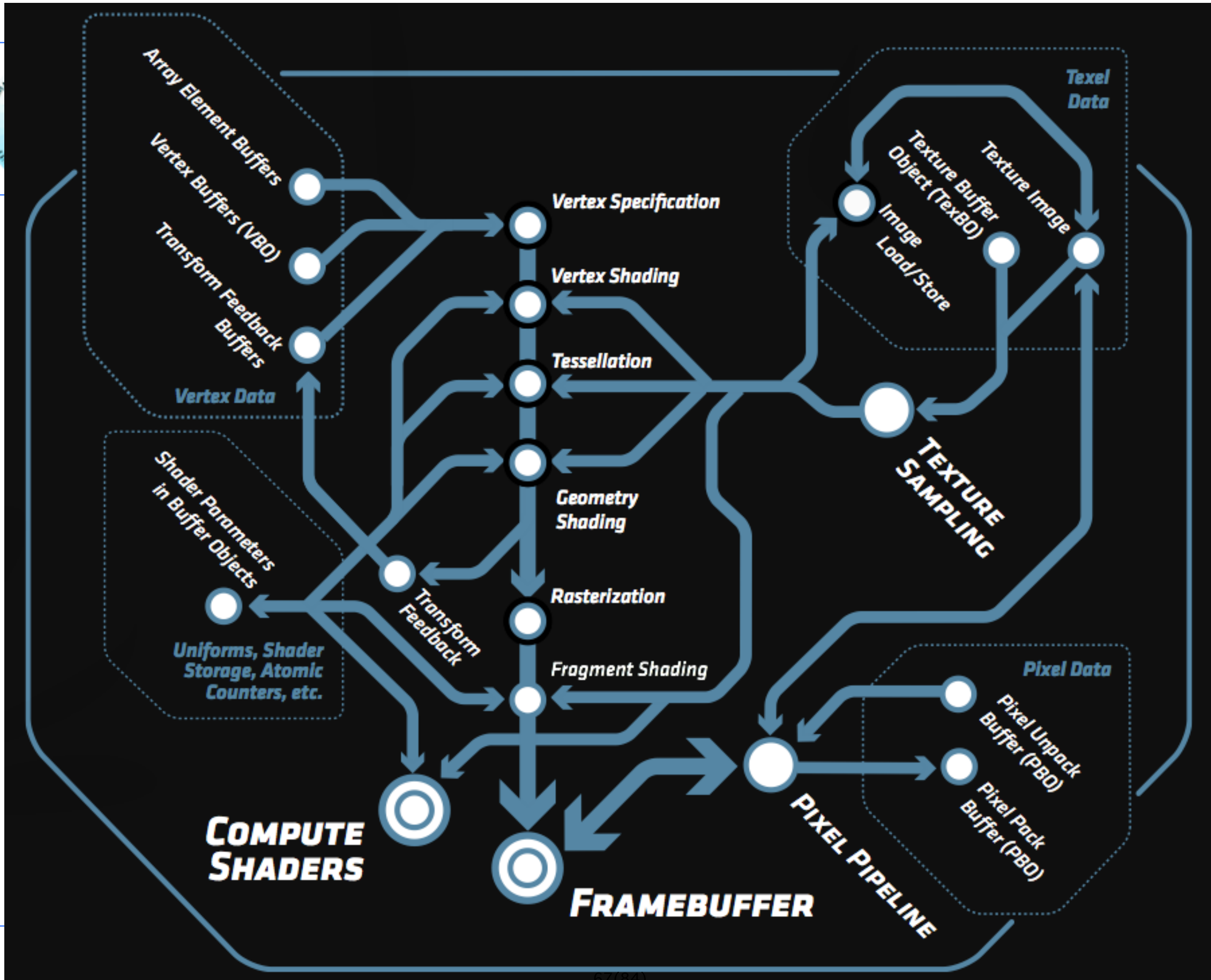
- + Bättre integration med OpenGL
- + Ingen extra installation behövs!
- + Enklare att konfigurera än OpenCL
- + Inte NVidia-specifikt som CUDA
- + Om du kan GLSL så är Compute Shaders (ganska) lätt!



Men det är ju inte bara plus...

- En del nya koncept
- Inte del av grafikpipelinen som fragment shaders
- Apple har egen lösning

Compute shaders är ensamma, kompileras inte med några andra.





OK, hur gör jag?

Kompileras som alla andra shaders!

Modifiera labbkoden från `GL_utilities`, kompilera (ensam) som `GL_COMPUTE_SHADER`.

Lätta saker:

- Uniforms fungerar som vanligt
- Texturer fungerar som vanligt



Information Coding / Computer Graphics, ISY, LiTH

Lite annorlunda än vanliga shaders

Attribut finns inte

Inte en tråd per fragment

Shader Storage Buffer Objects:

Generell buffertyp för godtyckliga data

Kan deklarerars så shadern ser det som en array av strukturer

Kan läsas och skrivas fritt av Compute Shaders!



Hur får jag in indata?

Ladda upp till SSBO:

```
glGenBuffers(1, &ssbo);  
glBindBuffer(GL_SHADER_STORAGE_BUFFER, ssbo);  
glBufferData(GL_SHADER_STORAGE_BUFFER, size, ptr,  
             GL_STATIC_DRAW);
```

Hur får shadern veta om den?

```
glBindBufferBase(GL_SHADER_STORAGE_BUFFER, id,  
                ssbo);
```

```
layout(std430, binding = id, buffer x {type y[]};
```



Hur accessar jag data i shadern?

Bestäm antal trådar per block:

```
layout(local_size_x = width, local_size_y = height)
```

Trådnummer:

```
gl_GlobalInvocation  
gl_LocalInvocation
```

```
void main()  
{  
    buffer[gl_GlobalInvocation.x] =  
        - buffer[gl_GlobalInvocation.x];  
}
```




Hur kör jag kärnan?

```
glUseProgram(program);
```

```
glDispatchCompute(sizex, sizey, sizez);
```

Argumenten till `glDispatchProgram` anger antalet block / workgroups. Antal trådar (work items) per block anges av shadern.



Hur får jag ut utdata?

```
glBindBuffer(GL_SHADER_STORAGE, ssbo);  
ptr = (int *) glMapBuffer(GL_SHADER_STORAGE,  
GL_READ_ONLY);
```

Läs sedan från ptr[i]

```
glUnmapBuffer(GL_SHADER_STORAGE);
```



Komplett huvudprogram:

```
int main(int argc, char **argv)
{
    glutInit (&argc, argv);
    glutCreateWindow("TEST1");

    // Load and compile the compute shader
    GLuint p =loadShader("cs.csh");

    GLuint ssbo; //Shader Storage Buffer Object

    // Some data
    int buf[16] = {1, 2, -3, 4, 5, -6, 7, 8, 9,
                  10, 11, 12, 13, 14, 15, 16};
    int *ptr;

    // Create buffer, upload data
    glGenBuffers(1, &ssbo);
    glBindBuffer(GL_SHADER_STORAGE_BUFFER, ssbo);
    glBufferData(GL_SHADER_STORAGE_BUFFER,
                16 * sizeof(int), &buf, GL_STATIC_DRAW);

    // Tell it where the input goes!
    // "5" matches "layout" in the shader.

    glBindBufferBase(GL_SHADER_STORAGE_BUFFER,
                    5, ssbo);

    // Get rolling!
    glDispatchCompute(16, 1, 1);

    // Get data back!
    glBindBuffer(GL_SHADER_STORAGE_BUFFER, ssbo);
    ptr = (int *)glMapBuffer(
        GL_SHADER_STORAGE_BUFFER,
        GL_READ_ONLY);
    for (int i=0; i < 16; i++)
    {
        printf("%d\n", ptr[i]);
    }
}
```



Enkel Compute Shader:

```
#version 430
#define width 16
#define height 16
```

OBS: Egentligen alldeles för mycket trådar för data (16*16*16)

```
// Compute shader invocations in each work group
```

```
layout(std430, binding = 5) buffer bbs {int bs[]};
```

```
layout(local_size_x=width, local_size_y=height) in;
```

```
//Kernel Program
```

```
void main()
```

```
{
```

```
    int i = int(gl_LocalInvocationID.x * 2);
```

```
    bs[gl_LocalInvocationID.x] = -bs[gl_LocalInvocationID.x];
```

```
}
```



Shared memory och synkronisering

Shared memory deklareraras *shared*.

```
shared float myShared[SIZE];
```

Synkronisering finns i flera former, kallas barriers.

```
barrier()  
memoryBarrier()  
memoryBarrierShared()  
groupMemoryBarrier()
```



Exempel: Transponering med shared memory

```
#version 450
#extension GL_ARB_compute_shader : enable
#define width 16
#define height 16

layout(std430, binding = 7) buffer bufc {float c[]};
layout(std430, binding = 5) buffer bufa {float a[]};
layout(local_size_x=width, local_size_y=height) in;

shared float s[width*height];

//Kernel Program
void main()
{
    uint theSizeX = gl_NumWorkGroups.x * gl_WorkGroupSize.x;
    uint theSizeY = gl_NumWorkGroups.y * gl_WorkGroupSize.y;

    int i = int(gl_GlobalInvocationID.y*theSizeX + gl_GlobalInvocationID.x);
    int j = int(gl_GlobalInvocationID.x*theSizeY + gl_GlobalInvocationID.y);
    int li = int(gl_LocalInvocationID.y*width + gl_LocalInvocationID.x);
    int lj = int(gl_LocalInvocationID.x*height + gl_LocalInvocationID.y);
    s[li] = a[lj];

    barrier();

    c[i] = s[lj];
}
```



Information Coding / Computer Graphics, ISY, LiTH

Liveexempel: Naiv matrismultiplikation

Bara naiva varianten.

Accelerering i stil med den jag fick med CUDA.



Information Coding / Computer Graphics, ISY, LiTH

Kan du köra Compute Shaders?

Krav: OpenGL 4.3 + Kepler!

Inget svårt problem. 600-serien och uppåt.

Men stöds inte av Apples OpenGL. (4.2)



Compute Shaders, ofta förbisett starkt alternativ

- Portabelt mellan olika grafikkort och OS
- I princip samma funktionalitet som CUDA och OpenCL
 - Ingen separat installation



Information Coding / Computer Graphics, ISY, LiTH

Projekt med CUDA eller Compute Shaders?

Varför inte - om du har ett problem stort nog.



RTX då?

Sedan NVidias 2000-serie!

RT-kärnor och Tensor cores

RT-kärnor: Snabbar upp tester mot enkla former (BVH)

Tensor-kärnor: Snabbar upp matrismultiplikationer

Finns som extensions för OpenGL, CUDA mm

`GL_NV_ray_tracing`

Fortfarande ont om exempel!



Information Coding / Computer Graphics, ISY, LiTH

Vågar jag använda RTX?

Verkar inte omöjligt... men anmärkningsvärt osäker mark
och jag har inte verifierat var extensions fungerar.

Projekt med osäkert resultat?

Helt OK och även små resultat kan räknas i sådana fall!



Information Coding / Computer Graphics, ISY, LiTH

Parallellprogrammering är framtiden!

All shaderprogrammering är parallellprogrammering.

Så gott som all prestandaökning i framtiden kommer från
parallelism.

Vi fortsätter i TDDD56.

...och kanske i era projekt i denna kurs?