Introduction to the GPGPU (General-purpose computing on graphics processing units) and to OpenCL

Prof. Dr.-Ing. Andreas Maier, Prof. Dr.-Ing. Dietmar Fey
J. Maier, B. Bier, A. Preuhs, C. Syben

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

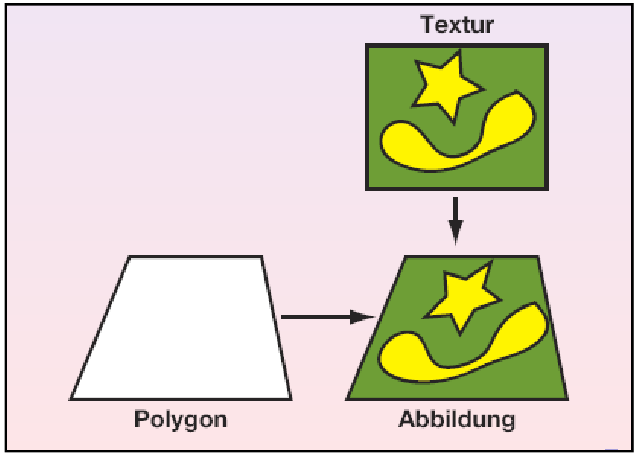
Outline

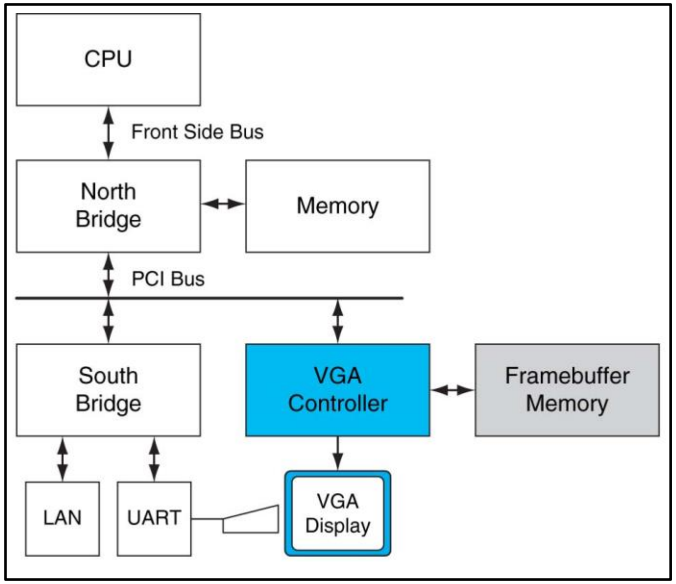
- Background and History
- OpenGL: Overview of the Graphic Pipeline
- GPU Architecture
- OpenCL
- OpenCL example

Background and History

GPGPU – General Purpose Graphics Processing Unit

- Graphics cards: Short background
  - Original use: Control of the PC‘s monitor by means of the graphics cards
  - Middle of the 80es: Graphics cards with 2D-acceleration
  - Beginning of the 90es: First 3D-Acceleration





GPGPU – Introduction (1)

- At the beginning, no standard programming interface was available
- Early 90s:
  - Establishment of OpenGL in the professional context
  - The fast development lead to successful spread in the market shares
- End of the 90s:
  - Graphics cards take over in the context of coordinate transformations and of Illumination (e.g. NVIDIA GeForce 256)
  - The term Graphics Processing Unit appears in the scientific community
- 2000s: Shader-Programming (Pixel-Shaders and Vertex-Shaders are used for graphic rendering)
- Nowadays:
  - Graphics cards manufacturers: ATI und NVIDIA
  - High demand on the market → low prices

GPGPU – Introduction (2)

Summary of the graphics cards‘ development:

VGA Controller
- Memory Controller
- Display Generator

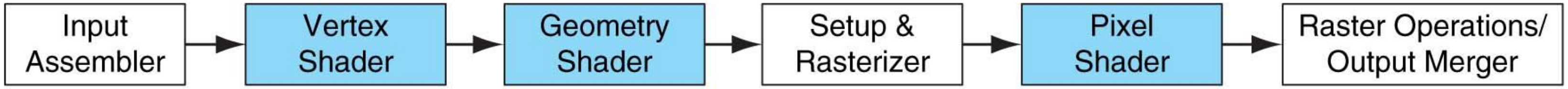
GPU (Graphics Processing Unit)
- The Processing of the traditional graphic pipeline

GPGPU (General Purpose Graphics Processing Unit)
- Programmable processors replaced fixed function blocks
- Increased computational accuracy
- Parallel programming is required → CUDA

OpenGL:
Overview of the Graphic Pipeline

Prof. Dr.-Ing. Andreas Maier (CS5), Prof. Dr. Ing. Dietmar Fey (CS3)

OpenGL – Graphic Pipeline



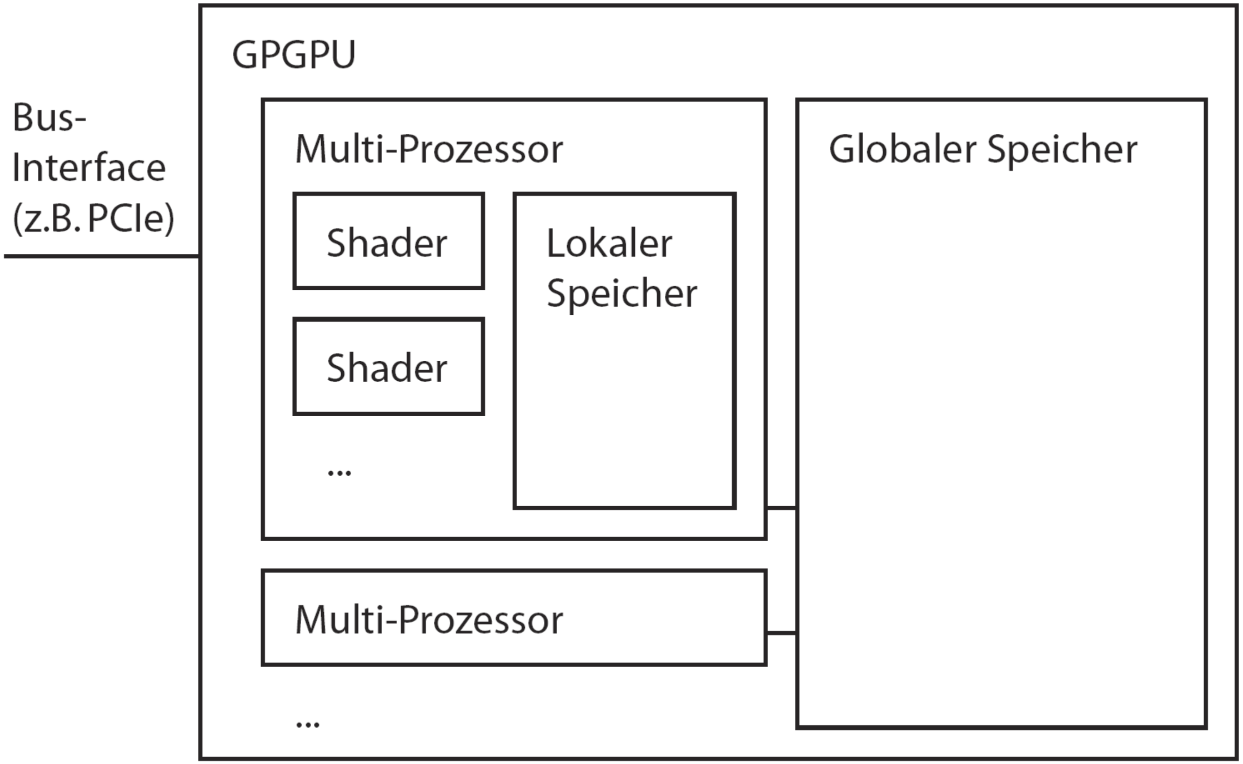
Input Assembler → Vertex Shader → Geometry Shader → Setup & Rasterizer → Pixel Shader → Raster Operations/ Output Merger

- Shader
  - It is the program in charge of the shadowing
  - It acts on junction points, (vertices), on geometric primitives (vertices, lines, triangles, etc.) and single points in the image
  - Some units are programmable (blue), some are hardwired (white)
- Textures
  - They describe characteristic of the points on a surface
  - Interpolation is done on floating points
  - Often disregarded in 1D, 2D- or 3D-fields

GPU Architecture

GPGPU – Architecture

- General configuration of a GPGPU



GPGPU – Memory Hierarchy on a GPU

- Global memory
  - Located in an external DRAM
  - Accessible just by a thread at a time
- Shared memory
  - Located in specific SRAM-banks
  - SM-specific
- Local memory
  - Thread-specific
  - Located in external DRAM
  - Configurable
- Special memory
  - Textures
  - Constants

OpenCL

The BIG idea behind OpenCL

- Loops are replaced by functions (*kernel*)
- *SIMT-Principle* - Single Instruction Multiple Threading

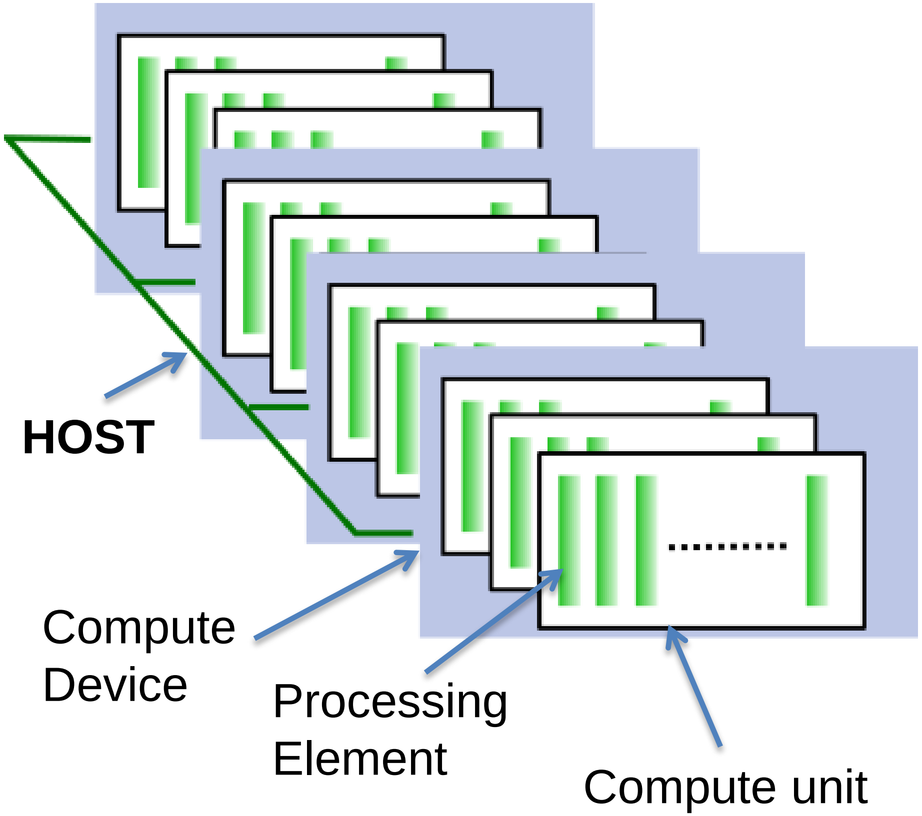
```
void
kernel_value_addition(int n,
                      float *a,
                      float value)
{
   for(int i = 0; i < n;i++)
       a[i] += value
}
```

```
void
kernel_value_addition(int n,
                        global float *a,
                        float value)
{
   int iGID = get_global_id(0);
   if(iGID >= n)
     return;

   a[iGID] += value
}
```

- Standard loop: The same code is repeated **serially**, one element after the other
- OpenCL kernel: The same code is executed at each point in the domain in **parallel**

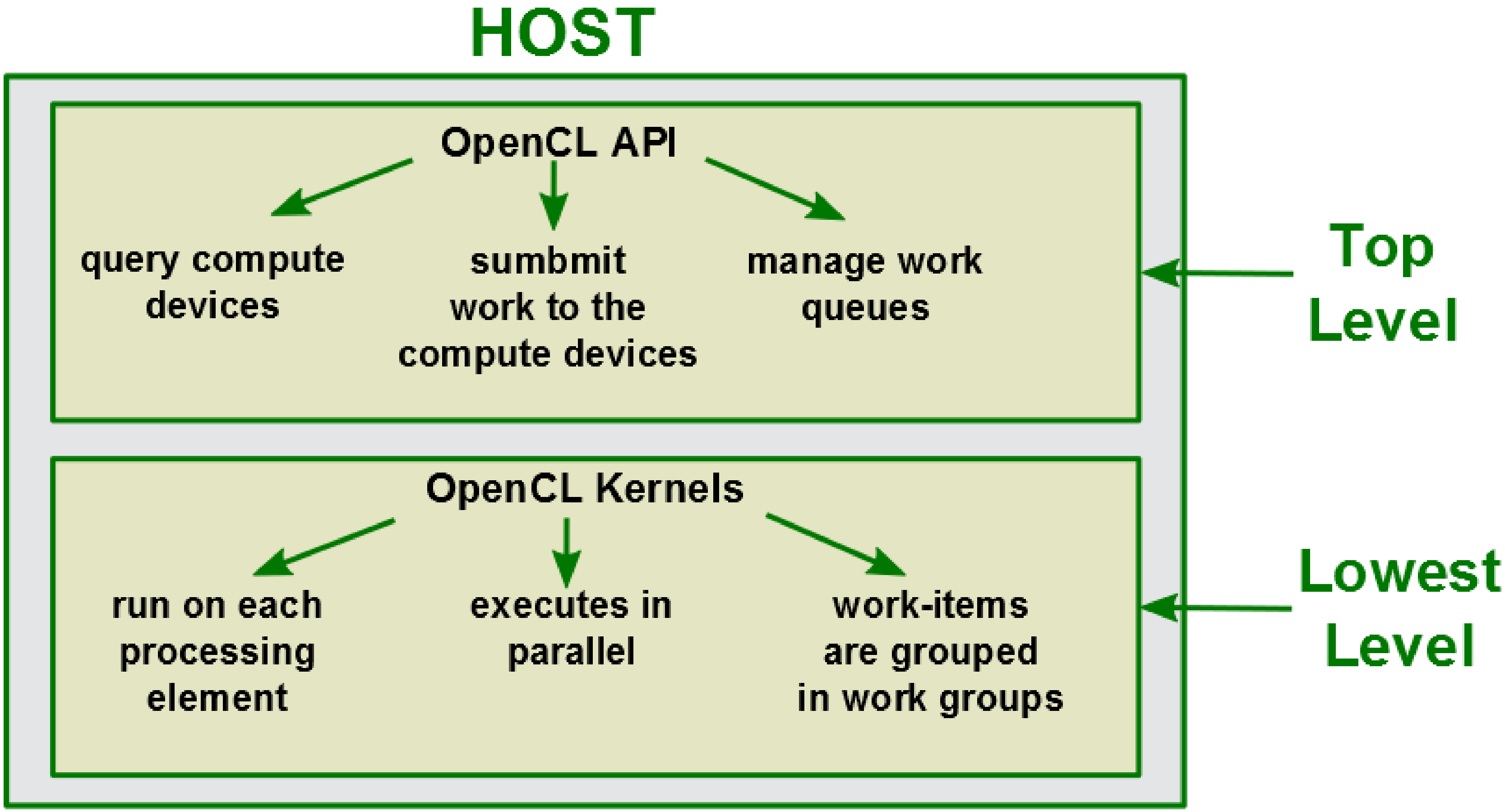
OpenCL (Open Computing Language)

- The OpenCL platform includes several components
  - One Host (CPU-based)
  - Compute Devices (GPU)
  - Compute units (groups of execution and arithmetic units)
  - Processing elements (executing OpenCL Kernels)
- *Serial code* executes in the Host
- *Parallel code* executes in the Compute devices



Woolley2011

OpenCL: How the execution works



HOST

OpenCL API

query compute devices

sumbmit work to the compute devices

manage work queues

Top Level

OpenCL Kernels

run on each processing element

executes in parallel

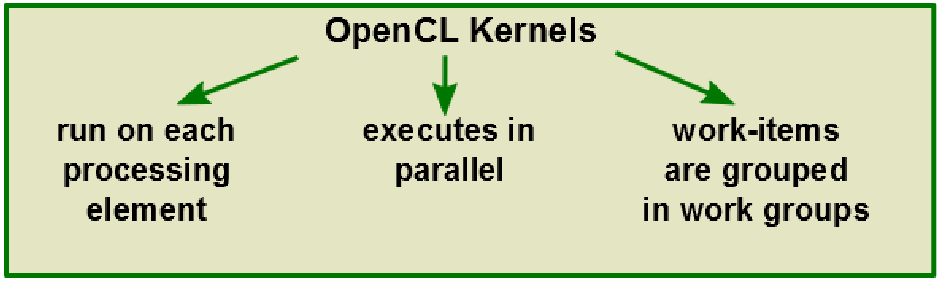
work-items are grouped in work groups

Lowest Level

Execution model

- Define a problem domain and execute a kernel invocation for each point in the domain:
  - An OpenCL kernel represents the code executed on a work item
  - Kernels process in parallel: their execution needs to be **independent** from one another
  - The code executed by the kernels needs to consider the required memory access

Lowest Level



OpenCL Kernels

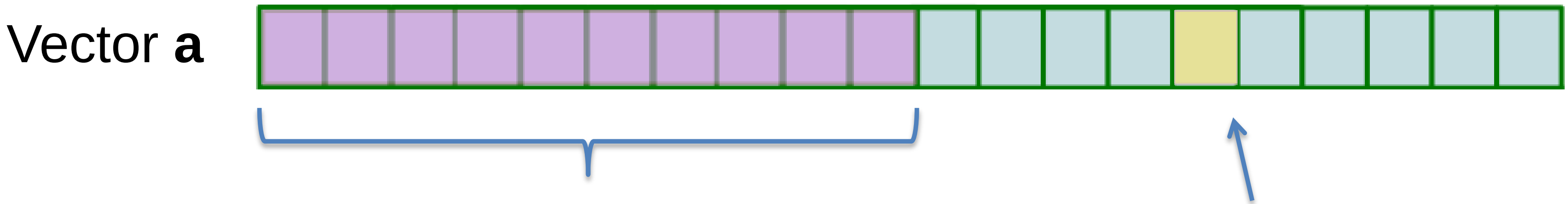
run on each processing element

executes in parallel

work-items are grouped in work groups

Work Domain

- The working domain consists of:
  - work items
  - working groups



Vector **a**

Working group

Working item

get_global_id = 14

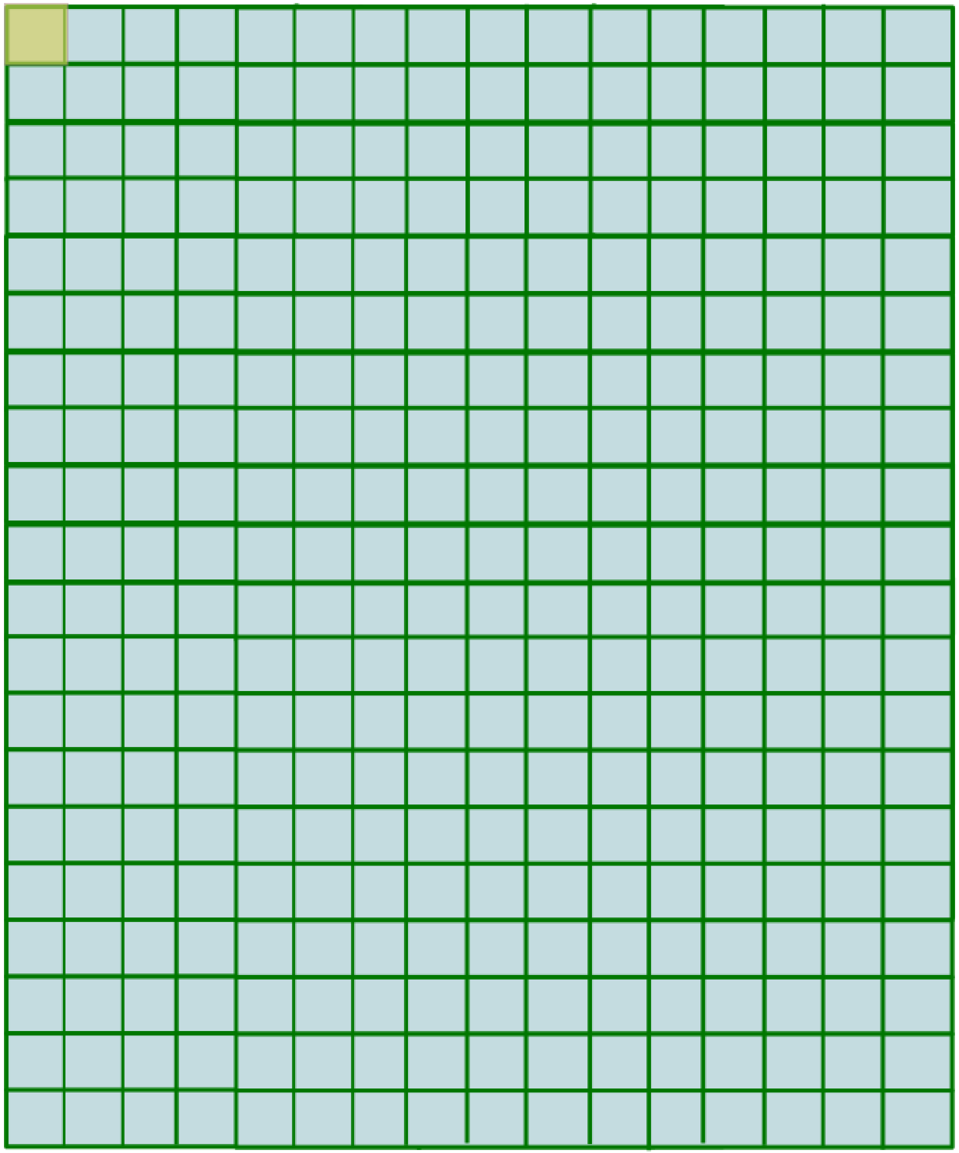
get_local_id = 4

get_global_size = 20

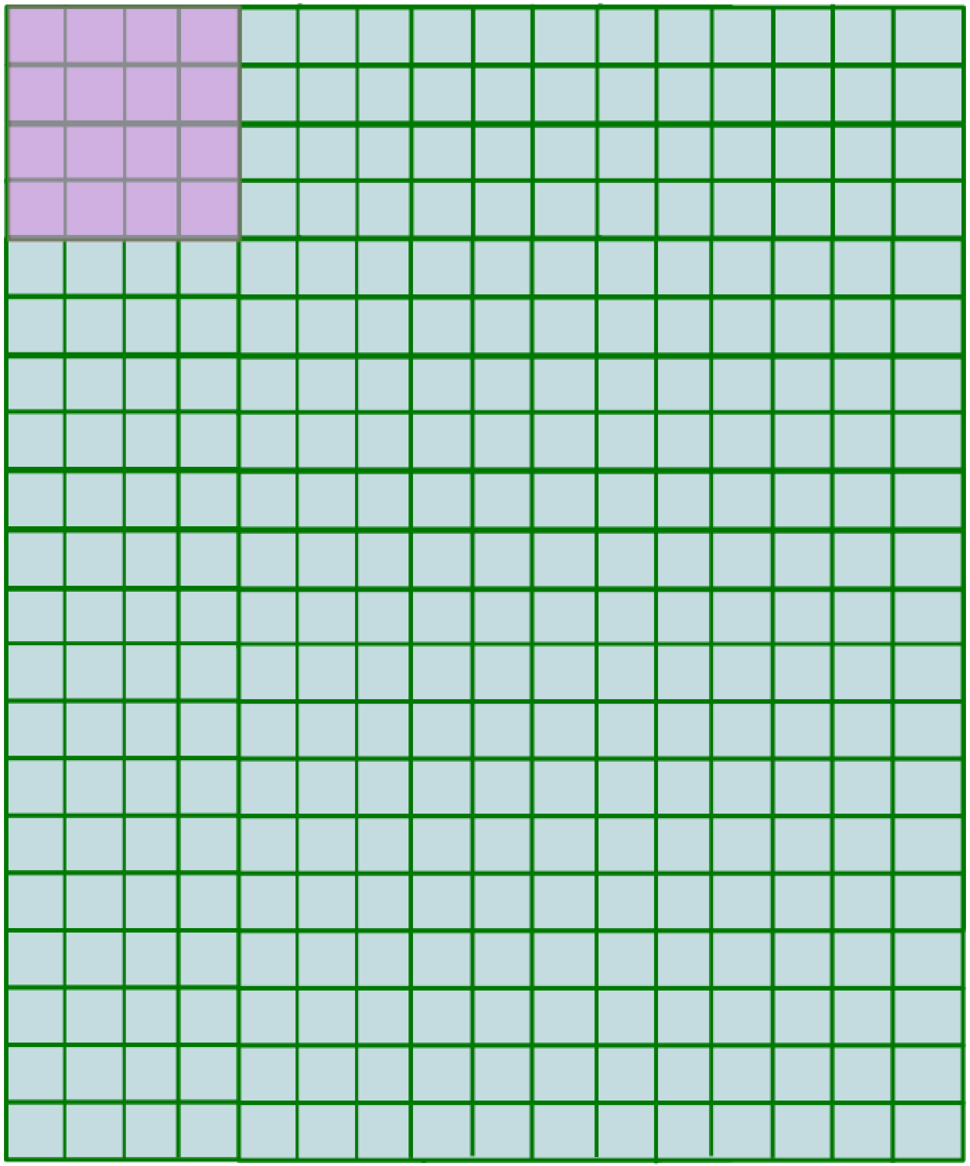
get_local_size =10

Example in 2D: Point-wise image processing

- The image represents the global dimension of the working domain (16 x 20)
- The global domain is divided into work items (e.g. a pixel)



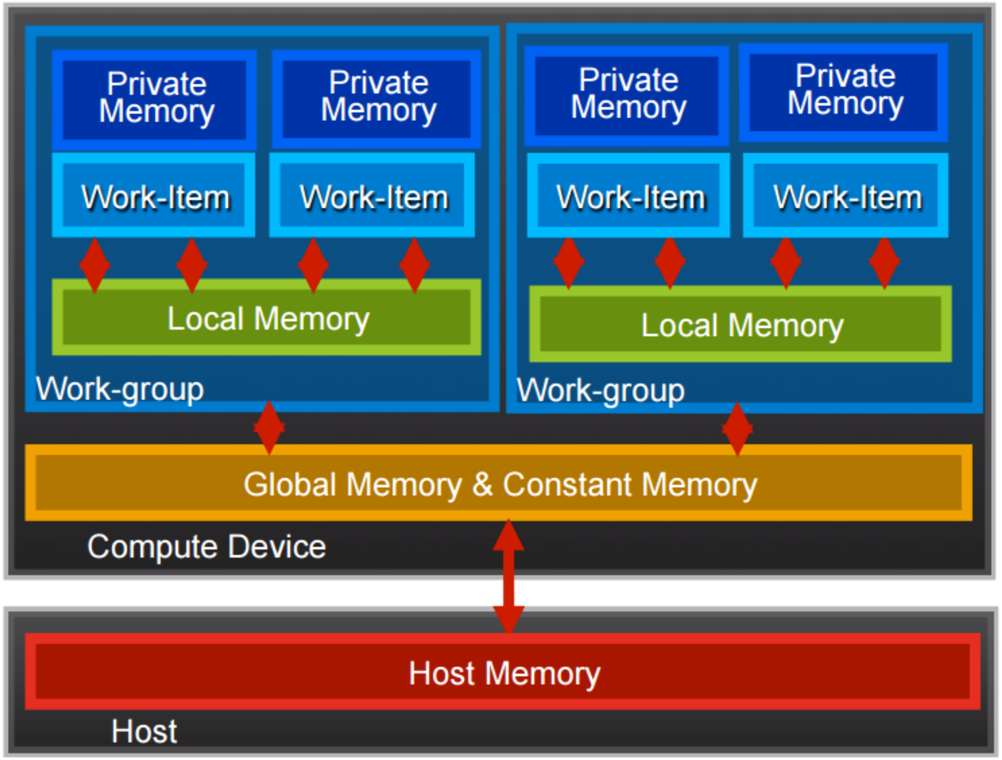
Example in 2D: Point-wise image processing



- The global domain is divided into work groups (4 x 4):
  - Work items are grouped in work groups which are executed together
- The work domain can be in practice bigger than the real size of the image
  - It needs to be a multiple of the worksize

OpenCL Memory Model

- Private Memory
  - Per work-item
- Local Memory
  - Shared within a work-group
- Global Memory
  - Visible to all work groups
- Host Memory
  - On the CPU



[Khronos2010]

OpenCL Example

Structure of the Host Program

- The host program is the code that runs on the host to setup the environment for the OpenCL program and manages the kernels

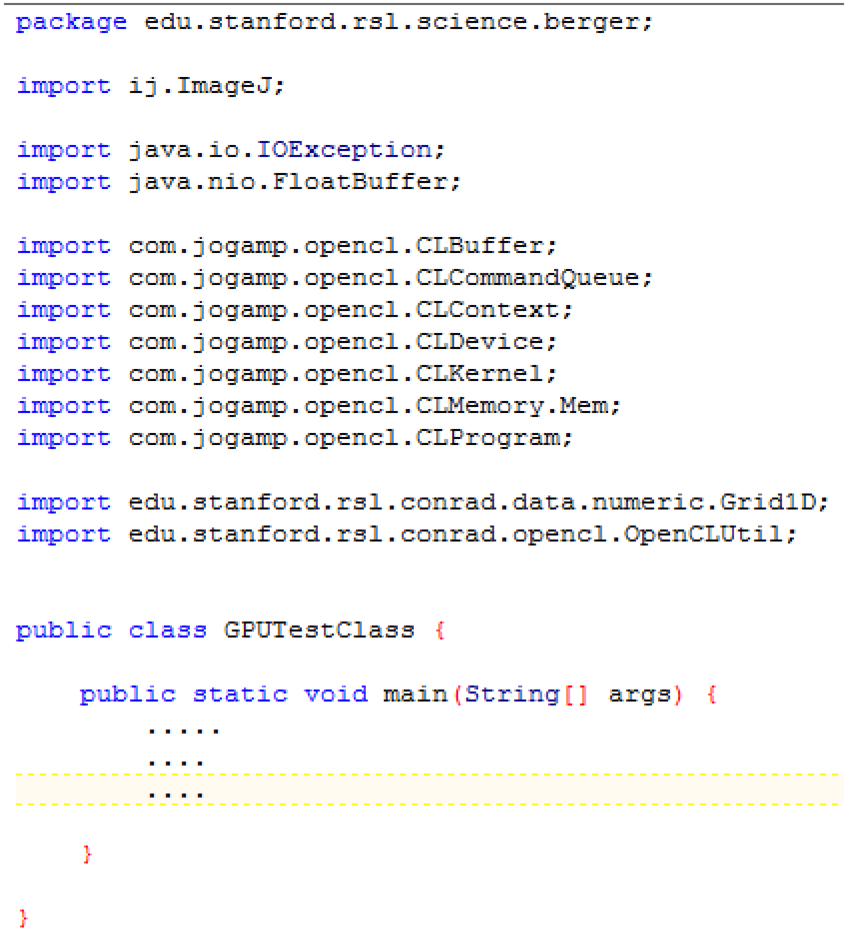
1. Define a **context**, **device** and **queues**

   Context: the environment within which kernels execute and in which synchronization and memory management is defined

   Device: the GPU

   Queues: all commands for the device are submitted through a queue
2. Create and build the **program**
3. Define **memory** objects
4. Define the **kernel**
5. Submit **commands** (transfer memory, execute kernel)

Example 1: Adding a value on a vector(1)



```java
package edu.stanford.rsl.science.berger;

import ij.ImageJ;

import java.io.IOException;
import java.nio.FloatBuffer;

import com.jogamp.opencl.CLBuffer;
import com.jogamp.opencl.CLCommandQueue;
import com.jogamp.opencl.CLContext;
import com.jogamp.opencl.CLDevice;
import com.jogamp.opencl.CLKernel;
import com.jogamp.opencl.CLMemory.Mem;
import com.jogamp.opencl.CLProgram;

import edu.stanford.rsl.conrad.data.numeric.Grid1D;
import edu.stanford.rsl.conrad.opencl.OpenCLUtil;


public class GPUTestClass {

    public static void main(String[] args) {
        .....
        ....
        ....

    }

}
```

- We want to write a class that uses our addition kernel:

```c
kernel void add(int n,
                global float *a,
                float value)
{
  int iGID = get_global_id(0);
  if(iGID >= n)
    return;

  a[iGID] += value
}
```

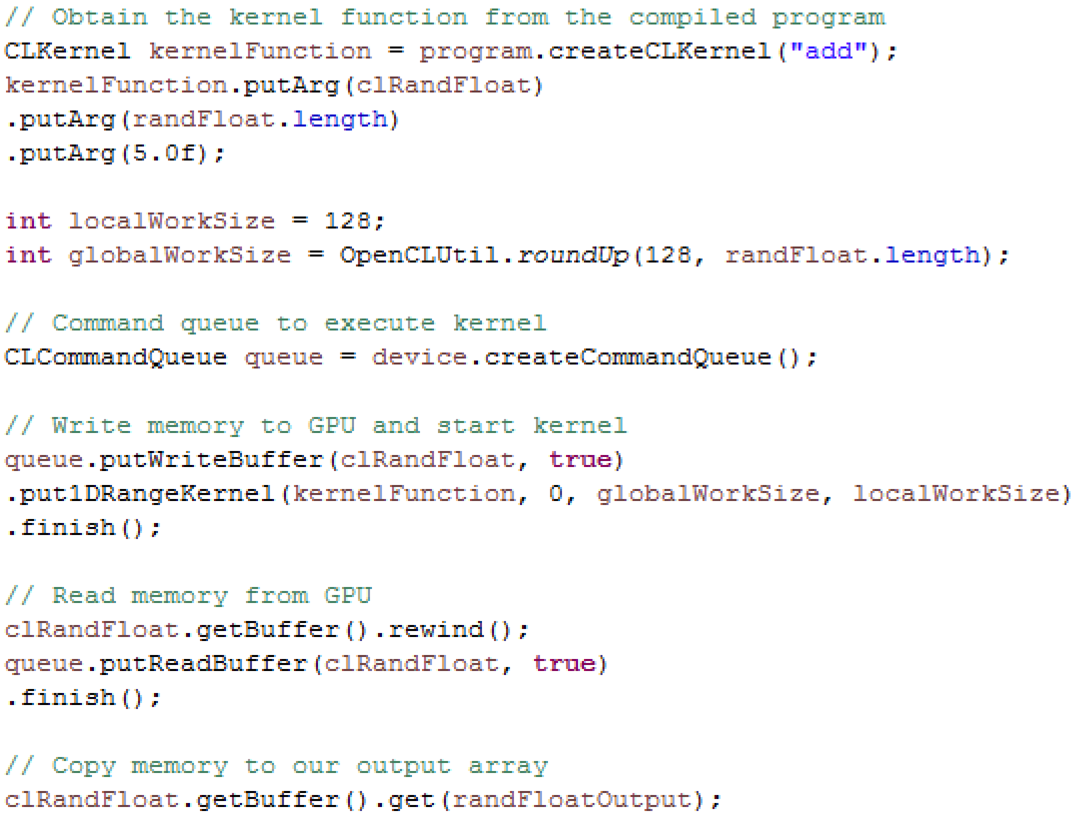
Code used with courtesy of Martin Berger

Example 1: Adding a value on a vector (2)

```
public class GPUTestClass {

    public static void main(String[] args) {

        float[] randFloat = new float[1024];
        for (int i = 0; i < randFloat.length; i++) {
            randFloat[i] = (float)(5*Math.random());
        }

        float[] randFloatOutput = new float[1024];

        // Create the context - Context used to obtain specific devices and to allocate GPU memory
        CLContext context = OpenCLUtil.getStaticContext();

        // Get the fastest device from context
        CLDevice device = context.getMaxFlopsDevice();

        // Load and compile the cl-Code that contains the kernel methods
        CLProgram program = null;
        try {
            program = context.createProgram(GPUTestClass.class.getResourceAsStream("VectorAdd.cl")).build();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        // Create a CLBuffer for the float array
        CLBuffer<FloatBuffer> clRandFloat = context.createFloatBuffer(randFloat.length, Mem.READ_WRITE);
        clRandFloat.getBuffer().put(randFloat);
        clRandFloat.getBuffer().rewind();
```

Code used with courtesy of Martin Berger

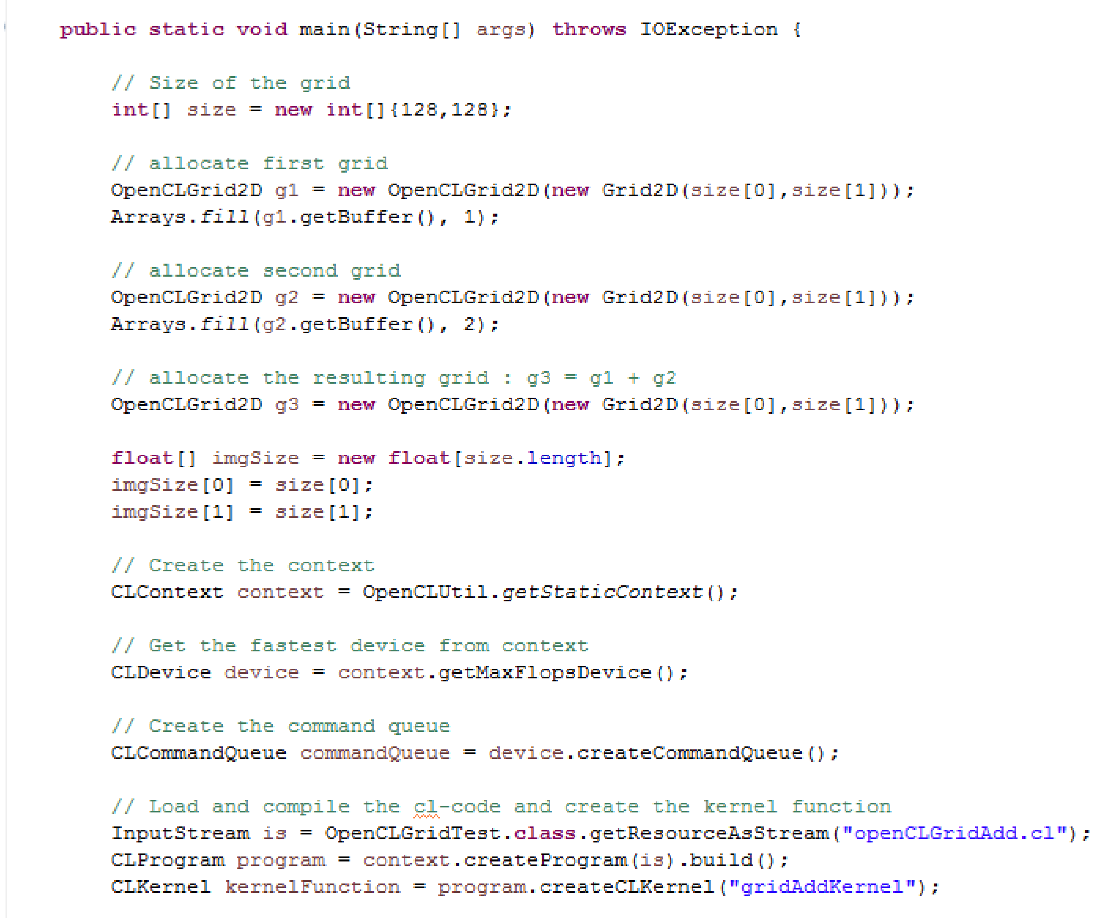
Example 1: Adding a value on a vector (3)



```
// Obtain the kernel function from the compiled program
CLKernel kernelFunction = program.createCLKernel("add");
kernelFunction.putArg(clRandFloat)
.putArg(randFloat.length)
.putArg(5.0f);

int localWorkSize = 128;
int globalWorkSize = OpenCLUtil.roundUp(128, randFloat.length);

// Command queue to execute kernel
CLCommandQueue queue = device.createCommandQueue();

// Write memory to GPU and start kernel
queue.putWriteBuffer(clRandFloat, true)
.put1DRangeKernel(kernelFunction, 0, globalWorkSize, localWorkSize)
.finish();

// Read memory from GPU
clRandFloat.getBuffer().rewind();
queue.putReadBuffer(clRandFloat, true)
.finish();

// Copy memory to our output array
clRandFloat.getBuffer().get(randFloatOutput);
```

Code used with courtesy of Martin Berger

Example 2: Adding two OpenCLGrid2D (Host Code, 1)



```
    public static void main(String[] args) throws IOException {

        // Size of the grid
        int[] size = new int[]{128,128};

        // allocate first grid
        OpenCLGrid2D g1 = new OpenCLGrid2D(new Grid2D(size[0],size[1]));
        Arrays.fill(g1.getBuffer(), 1);

        // allocate second grid
        OpenCLGrid2D g2 = new OpenCLGrid2D(new Grid2D(size[0],size[1]));
        Arrays.fill(g2.getBuffer(), 2);

        // allocate the resulting grid : g3 = g1 + g2
        OpenCLGrid2D g3 = new OpenCLGrid2D(new Grid2D(size[0],size[1]));

        float[] imgSize = new float[size.length];
        imgSize[0] = size[0];
        imgSize[1] = size[1];

        // Create the context
        CLContext context = OpenCLUtil.getStaticContext();

        // Get the fastest device from context
        CLDevice device = context.getMaxFlopsDevice();

        // Create the command queue
        CLCommandQueue commandQueue = device.createCommandQueue();

        // Load and compile the cl-code and create the kernel function
        InputStream is = OpenCLGridTest.class.getResourceAsStream("openCLGridAdd.cl");
        CLProgram program = context.createProgram(is).build();
        CLKernel kernelFunction = program.createCLKernel("gridAddKernel");
```

Code used with courtesy of Mathias Unberath

Example 2: Adding two OpenCLGrid2D (Host Code, 2)

```
// Create the OpenCL Grids and set their texture

// Grid 1
CLBuffer<FloatBuffer> gImgSize = context.createFloatBuffer(imgSize.length, Mem.READ_ONLY);
gImgSize.getBuffer().put(imgSize);
gImgSize.getBuffer().rewind();

// Create the CLBuffer for the grids
CLImageFormat format = new CLImageFormat(ChannelOrder.INTENSITY, ChannelType.FLOAT);

// make sure OpenCL is turned on / and things are on the device
g1.getDelegate().prepareForDeviceOperation();
g1.getDelegate().getCLBuffer().getBuffer().rewind();

// Create and set the texture
CLImage2d<FloatBuffer> g1Tex = null;
g1Tex = context.createImage2d(g1.getDelegate().getCLBuffer().getBuffer(), size[0], size[1], format, Mem.READ_ONLY);
g1.getDelegate().release();

// Grid 2
g2.getDelegate().prepareForDeviceOperation();
g2.getDelegate().getCLBuffer().getBuffer().rewind();

// Create and set the texture image for second grid
CLImage2d<FloatBuffer> g2Tex = null;
g2Tex = context.createImage2d(g2.getDelegate().getCLBuffer().getBuffer(),size[0], size[1], format, Mem.READ_ONLY);
g2.getDelegate().release();

// Grid 3
g3.getDelegate().prepareForDeviceOperation();
```

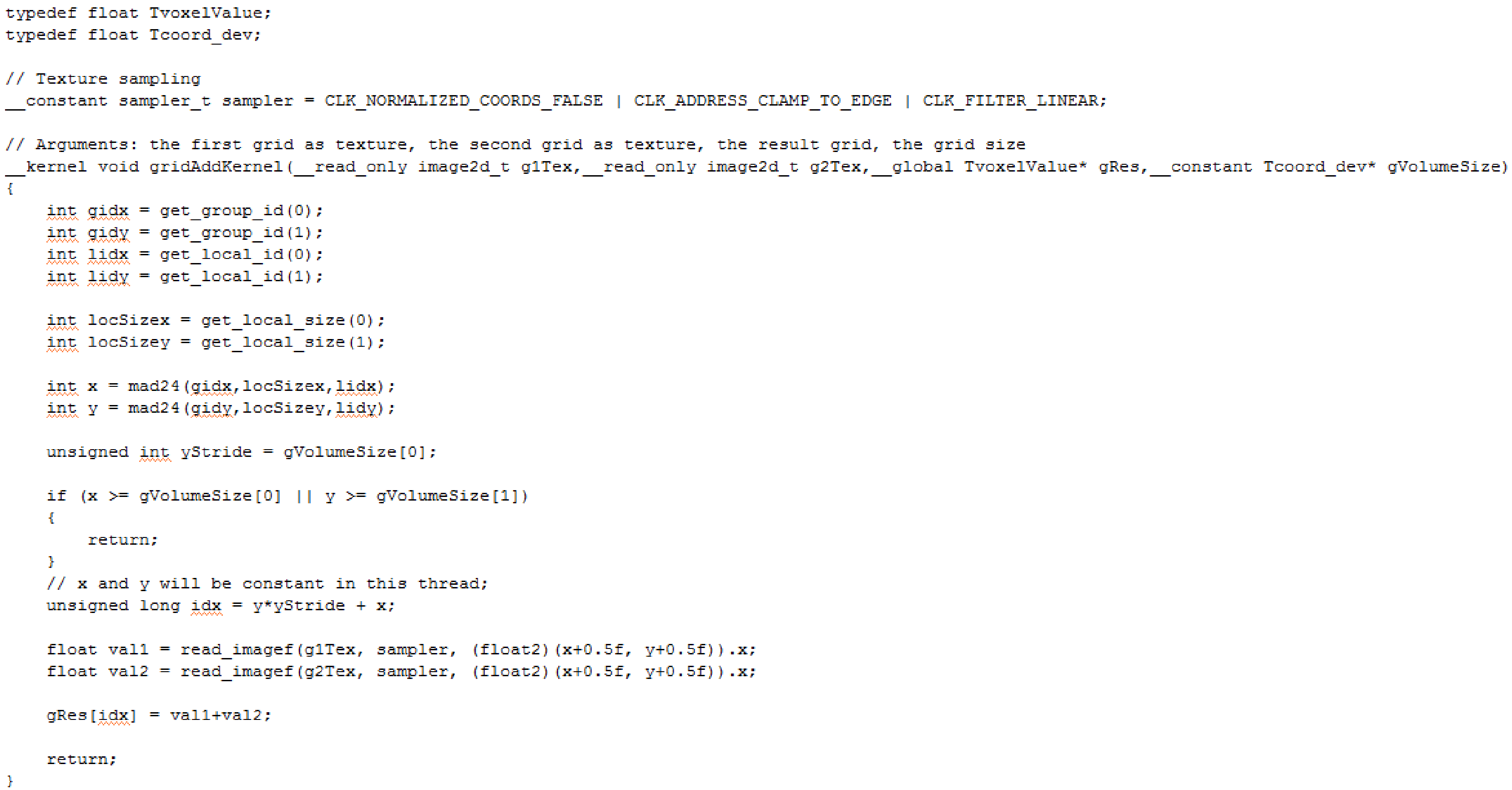
Code used with courtesy of Mathias Unberath

Example 2: Adding two OpenCLGrid2D (Host Code, 3)

```
        // Write memory on the GPU
        commandQueue
        .putWriteImage(g1Tex, true) // writes the first texture
        .putWriteImage(g2Tex, true) // writes the second textue
        .putWriteBuffer(g3.getDelegate().getCLBuffer(), true) // writes the third image buffer
        .putWriteBuffer(gImgSize,true)
        .finish();

        // Write kernel parameters
        kernelFunction.rewind();
        kernelFunction
        .putArg(g1Tex)
        .putArg(g2Tex)
        .putArg(g3.getDelegate().getCLBuffer())
        .putArg(gImgSize);

        // Check correct work group sizes
        int bpBlockSize[] = {32, 32};
        int maxWorkGroupSize = device.getMaxWorkGroupSize();
        int[] realLocalSize = new int[]{    Math.min((int)Math.pow(maxWorkGroupSize,1/2.0), bpBlockSize[0]),
                Math.min((int)Math.pow(maxWorkGroupSize,1/2.0), bpBlockSize[1])};

        // rounded up to the nearest multiple of localWorkSize
        int[] globalWorkSize = new int[]{OpenCLUtil.roundUp(realLocalSize[0], (int)imgSize[0]),
                OpenCLUtil.roundUp(realLocalSize[1], (int)imgSize[1])};

        // execute kernel
        commandQueue.put2DRangeKernel(kernelFunction, 0, 0, globalWorkSize[0], globalWorkSize[1],
                realLocalSize[0], realLocalSize[1]).finish();
        g3.getDelegate().notifyDeviceChange();

        new ImageJ();
        new Grid2D(g3).show();
    }
}
```

Code used with courtesy of Mathias Unberath

Example 2: Adding two OpenCLGrid2D (Kernel Code)



```
typedef float TvoxelValue;
typedef float Tcoord_dev;

// Texture sampling
__constant sampler_t sampler = CLK_NORMALIZED_COORDS_FALSE | CLK_ADDRESS_CLAMP_TO_EDGE | CLK_FILTER_LINEAR;

// Arguments: the first grid as texture, the second grid as texture, the result grid, the grid size
__kernel void gridAddKernel(__read_only image2d_t g1Tex,__read_only image2d_t g2Tex,__global TvoxelValue* gRes,__constant Tcoord_dev* gVolumeSize)
{
    int gidx = get_group_id(0);
    int gidy = get_group_id(1);
    int lidx = get_local_id(0);
    int lidy = get_local_id(1);

    int locSizex = get_local_size(0);
    int locSizey = get_local_size(1);

    int x = mad24(gidx,locSizex,lidx);
    int y = mad24(gidy,locSizey,lidy);

    unsigned int yStride = gVolumeSize[0];

    if (x >= gVolumeSize[0] || y >= gVolumeSize[1])
    {
        return;
    }
    // x and y will be constant in this thread;
    unsigned long idx = y*yStride + x;

    float val1 = read_imagef(g1Tex, sampler, (float2)(x+0.5f, y+0.5f)).x;
    float val2 = read_imagef(g2Tex, sampler, (float2)(x+0.5f, y+0.5f)).x;

    gRes[idx] = val1+val2;

    return;
}
```

Code used with courtesy of Mathias Unberath

References

- **Woolley2011**: http://www.cc.gatech.edu/~vetter/keeneland/tutorial-2011-04-14/06-intro_to_opencl.pdf
- **Khronos2010**: https://www.khronos.org/assets/uploads/developers/library/overview/OpenCL-Overview-Jun10.pdf
- **Smith2013**:

https://www.cs.bris.ac.uk/home/simonm/SC13/OpenCL_slides_SC13.pdf

http://www.cs.bris.ac.uk/home/simonm/workshops/opencl_2day.html

Questions?