Χορεύοντας με τα teraflops

προγραμματισμός στις σύγχρονες παράλληλες αρχιτεκτονικές

Γιάννης Τσιομπίκας

nuclear@member.fsf.org

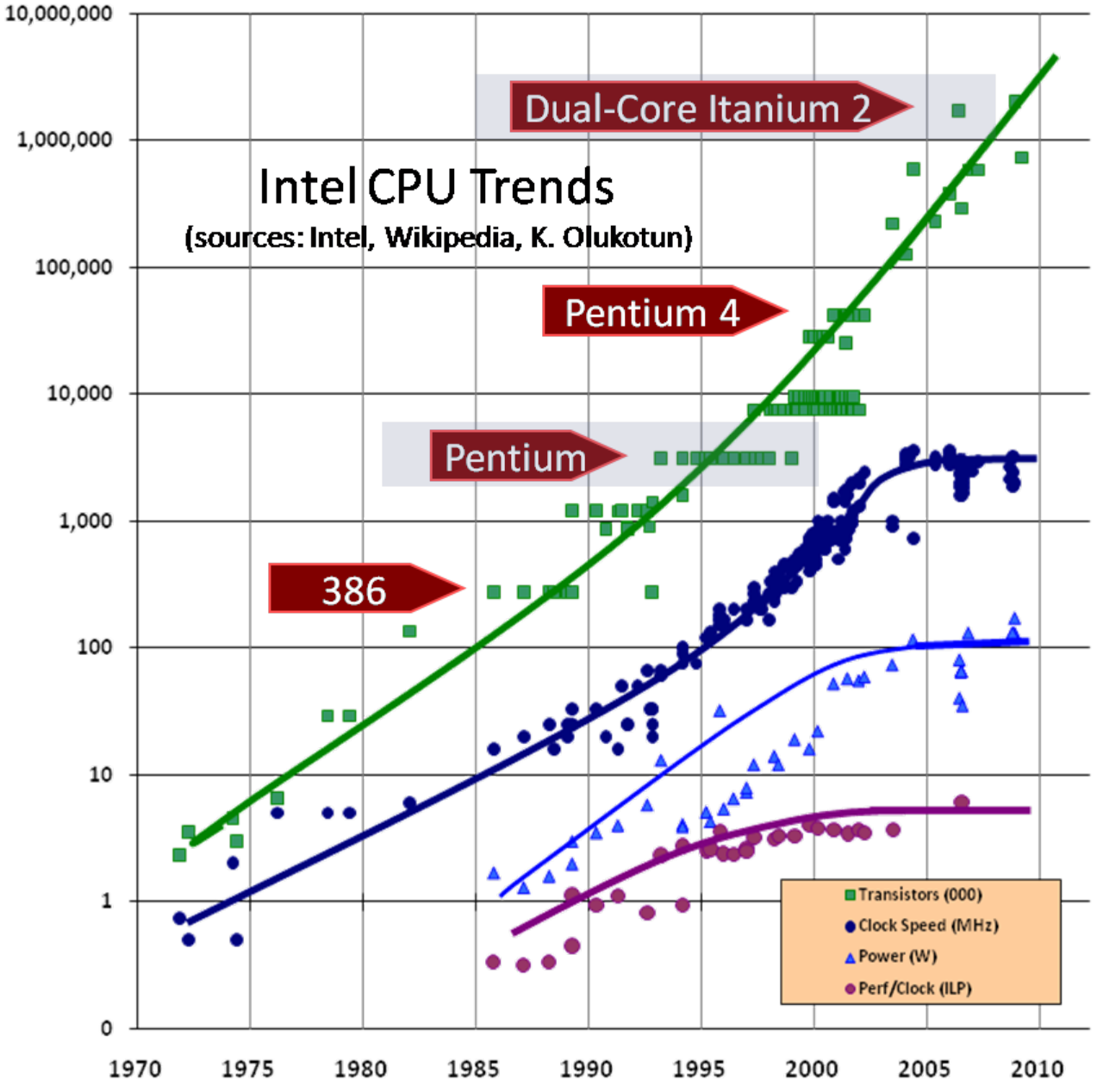
21 Απριλίου 2013

Εισαγωγή

CPU speed - Παρελθόν

- Εκθετική αύξηση της υπολογιστικής υσχύος τον επεξεργαστών.
- Moore's law
- Αγώνας δρόμου των MHz

CPU speed - Παρόν

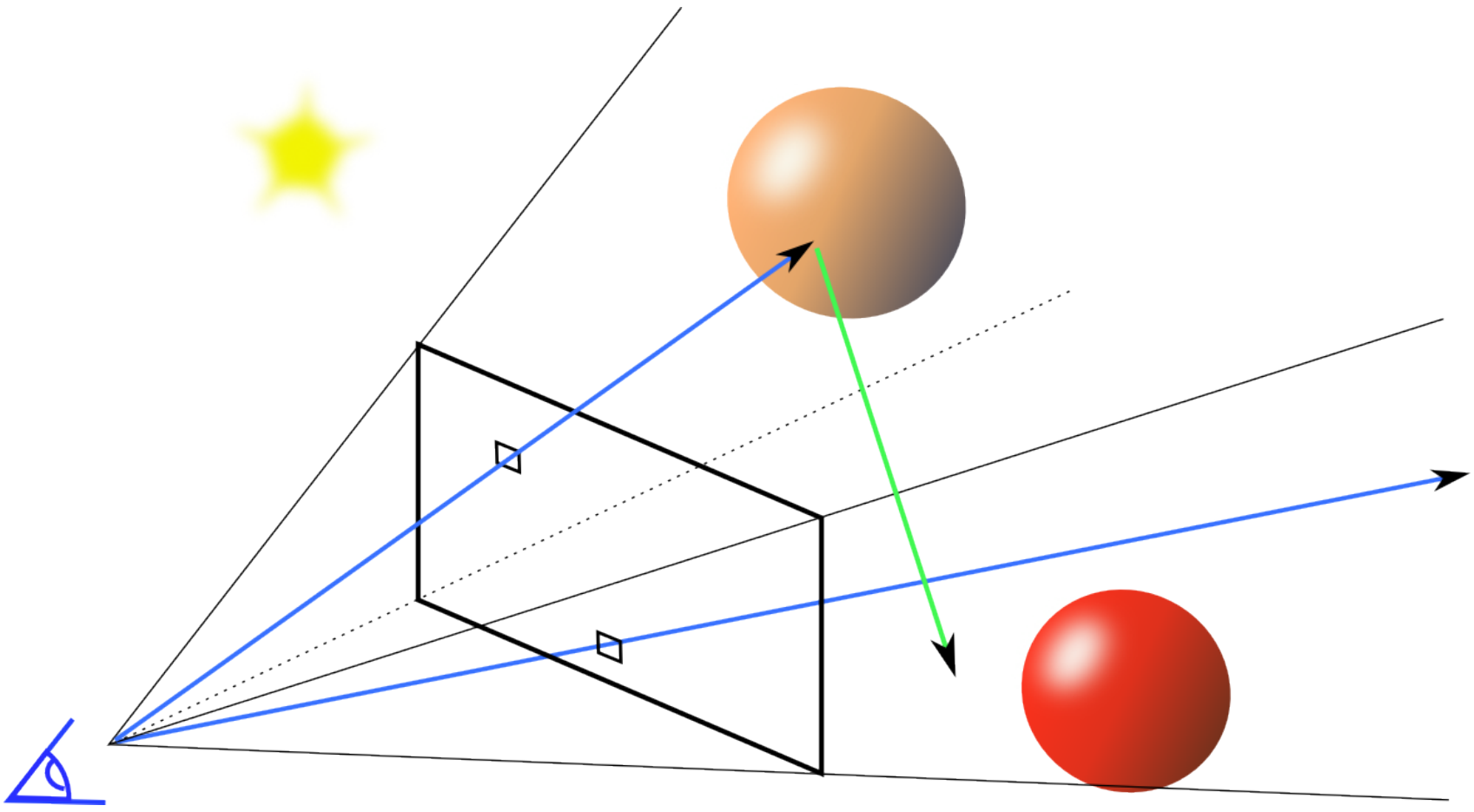


Παραλληλησμός

- Vectorization (SIMD instructions)
- SMP (Symmetric Multi-Processing)
  - Multiple processors
  - Multiple processor cores
- Heterogenous computing
  - special processors (π.χ. CELL SPEs)
  - Graphics processors (GPU)
- Distributed computing

Παράδειγμα

ray tracer



```
for(int i=0; i<frame->ysz; i++) {
    Color *pixel = frame->pixels + i * frame->xsz;
    for(int j=0; j<frame->xsz; j++) {
        Ray ray = camera->get_primary_ray(j, i);
        *pixel++ = trace_ray(scn, ray);
    }
}
```

- `trace_ray`: Βρίσκει πού "χτυπάει" μια ακτίνα, και καλεί την `shade` για να υπολογίσει χρώμα σε αυτό το σημείο, το οποίο επιστρέφει.
- `shade`: Κάνει υπολογισμούς φωτισμού για να βρεί το χρώμα σε κάποιο σημείο και το επιστρέφει. Μπορεί να καλέσει την `trace_ray`, για να συλλέξει φωτισμό και απο άλλες κατευθύνσεις.

Περιγραφή σκηνής

```
environment -color 0.2 0.25 0.3 -texture clouds.cubemap

material -name foo -diffuse 1 0.2 0.1 -specular 1 1 1 -shininess 60
material -name gnd -reflect 0.3 -texture tiles.jpg

sphere -name sph -center 0 0 0 -radius 1 -material foo
plane -name floor -normal 0 1 0 -distance -1 -material gnd

light -position -10 10 -20 -color 0.8 0.8 0.8

xform -name sph -time 0 -pos 0 1 0
xform -name sph -time 1 -pos 0 0 0
```

Single-threaded raytracer example video

Symmetric Multi-Processing

Symmetric Multi-Processing

Κάθε επεξεργαστής ή core εκτελεί ένα Kernel Schedulable Entity ανα πάσα στιγμή.

Processes

- `fork` (UNIX)
- `CreateProcess` (Windows)

Threads

- POSIX threads (`pthread_create`)
- C11 threads (`thrd_create`)
- C++11 threads (`std::thread`, `std::async`, `std::promise`/`std::future`)
- OpenMP (`#pragma omp parallel`)

Multithreading

- Διαχωρισμός του υπολογισμού σε πολλαπλά threads
  - Στατικός διαχωρισμός (π.χ. 4 threads - 1/4 του frame το καθ'ένα)
  - Worker thread pool
- Συγχρονισμός
  - Mutual exclusion: phtread_mutex_lock, mtx_lock, std::mutex::lock, std::unique_lock
  - Condition variables: pthread_cond_wait/pthread_cond_signal/pthread_cond_broadcast, cnd_wait/cnd_signal/cnd_broadcast, std::condition_variable
  - pthread_join/pthread_detatch, thrd_join/thrd_detatch, std::thread::join, std::thread::detatch.

Worker thread pool

- n worker threads κοιμούνται με `pthread_cond_wait` στο `work_pending` condvar.
- Το main thread:
  - Κλειδώνει το mutex
  - Προσθέτει work list items
  - Ξυπνάει τους workers με `pthread_cond_broadcast`
  - Ξεκλειδώνει το mutex
- Το worker thread:
  - Κλειδώνει το mutex
  - Αν υπάρχει δουλειά στη λίστα τήν αφαιρεί, και την εκτελεί (αφού αφήσει το mutex).
  - Αν δεν υπάρχει δουλειά στη λίστα ξανα-πέφτει για ύπνο κάνοντας wait στο condvar.

OpenMP

- High-level parallelization
- Αυτοματοποιεί τον παραλληλισμό τμημάτων του κώδικα που θέλουμε να εκτελεστούν παράλληλα.
- Απαιτεί υποστίριξη απο τον compiler (ενεργοποιείται στον gcc με `-fopenmp`)
- Ο compiler κάνει emit κλήσεις στο OpenMP runtime library που υλοποιεί worker threads, synchronization εργαλεία, κ.α. (στον gcc: `-lgomp`)

OpenMP

```
for(int i=0; i<frame->ysz; i++) {
    Color *pixel = frame->pixels + i * frame->xsz;
    for(int j=0; j<frame->xsz; j++) {
        Ray ray = camera->get_primary_ray(j, i);
        *pixel++ = trace_ray(scn, ray);
    }
}
```

OpenMP

```
#pragma omp parallel for
for(int i=0; i<frame->ysz; i++) {
    Color *pixel = frame->pixels + i * frame->xsz;
    for(int j=0; j<frame->xsz; j++) {
        Ray ray = camera->get_primary_ray(j, i);
        *pixel++ = trace_ray(scn, ray);
    }
}
```

Multi-threaded raytracer example video

GPU Computing

Εξέλιξη των PC καρτών γραφικών

- 80s: CGA, EGA, VGA framebuffer
- αρχές-μέσα 90: SuperVGA framebuffer
- τέλη 90 - αρχές 2000: 3dfx, ATI, NVIDIA "3D accelerators"
- αρχές-μέσα 2000: programmable shaders (Geforce3) τραγικά περιορισμένο υπολογιστικό μοντέλο.
- τέλη 2000 - σήμερα: Geforce8800+ πλήρως προγραμματιζόμενες GPU με πολλούς πυρήνες εκτέλεσης.

GPU programing languages

Shader assembly dialects

- OpenGL ARB vertex/pixel programs
- Direct3D vertex/pixel shaders 2.0

High-level shading languages

- nvidia Cg (C for graphics)
- OpenGL Shading Language (GLSL)
- DirectX HLSL

GPGPU computing languages

- nvidia cuda
- OpenCL (Open Computing Language)
- OpenGL Compute Shaders

GPU programing languages

Shader assembly dialects

- OpenGL ARB vertex/pixel programs
- Direct3D vertex/pixel shaders 2.0

High-level shading languages

- nvidia Cg (C for graphics)
- OpenGL Shading Language (GLSL)
- DirectX HLSL

GPGPU computing languages

- nvidia cuda
- OpenCL (Open Computing Language)
- OpenGL Compute Shaders

GPU programing languages

Shader assembly dialects

- OpenGL ARB vertex/pixel programs
- Direct3D vertex/pixel shaders 2.0

High-level shading languages

- nvidia Cg (C for graphics)
- OpenGL Shading Language (GLSL)
- DirectX HLSL

GPGPU computing languages

- nvidia cuda
- OpenCL (Open Computing Language)
- OpenGL Compute Shaders

GPU programing languages

Shader assembly dialects

- OpenGL ARB vertex/pixel programs
- Direct3D vertex/pixel shaders 2.0

High-level shading languages

- nvidia Cg (C for graphics)
- OpenGL Shading Language (GLSL)
- DirectX HLSL

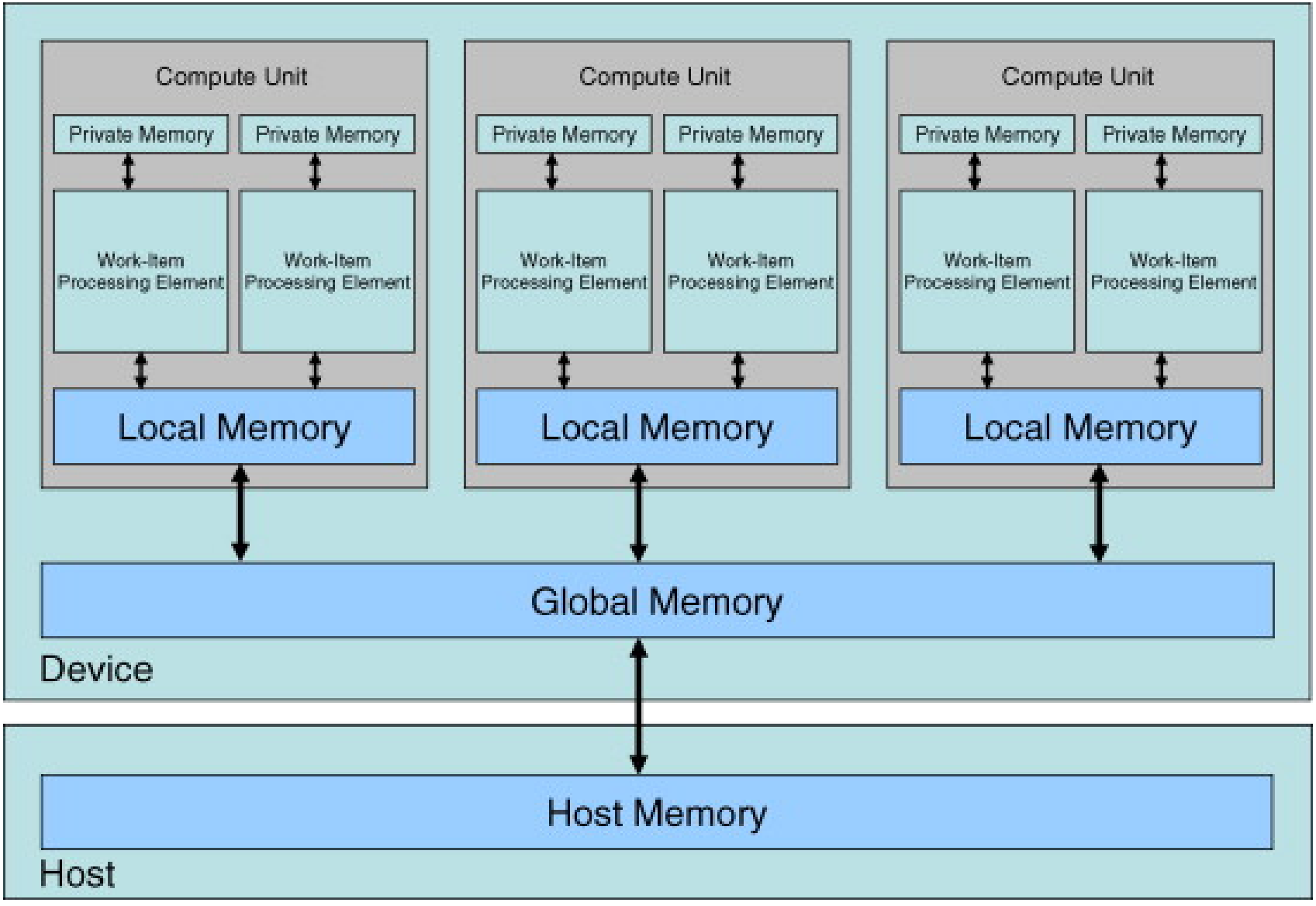
GPGPU computing languages

- nvidia cuda
- OpenCL (Open Computing Language)
- OpenGL Compute Shaders

GPU computing architecture

- Multiple floating point processors
- Groups of processors (warps) με κοινή local μνήμη και *κοινό execution control*
- Δυναμικό scheduling των groups με αμελητέο context switching overhead
- Ιεραρχία μνήμης (επόμενο slide)

GPU memory hierarchy



OpenCL vs CUDA vs GLSL

OpenCL	CUDA	GLSL
work item	thread	fragment
work group	thrad block	N/A
global memory	global memory	OpenGL textures & buffers
constant memory	constant memory	uniforms
local memory	shared memory	N/A
private memory	registers	local variables
get_global_id(0)	threadIdx.x + blockIdx.x * blockDim.x	gl_FragCoord.x

Απλό παράδειγμα OpenCL

Host code:

```
inbuf = clCreateBuffer(ctx, CL_MEM_READ_ONLY, sz, 0, &err);
outbuf = clCreateBuffer(ctx, CL_MEM_WRITE_ONLY, sz, 0, &err);

clEnqueueWriteBuffer(cmdq, inbuf, 1, 0, sz, data, 0, 0, 0);

prog = clCreateProgramWithSource(ctx, 1, &src_buf, 0, &err);
clBuildProgram(prog, 0, 0, 0, 0, 0);
kernel = clCreateKernel(prog, "square", &err);
clSetKernelArg(kernel, 0, sz, &inbuf);
clSetKernelArg(kernel, 1, sz, &outbuf);

int globsz[] = {1, 0, 0};
clEnqueueNDRangeKernel(cmdq, kernel, 1, 0, globsz, 0, 0, 0, 0);

clEnqueueReadBuffer(cmdq, outbuf, 1, 0, sz, data, 0, 0, 0, 0);
```

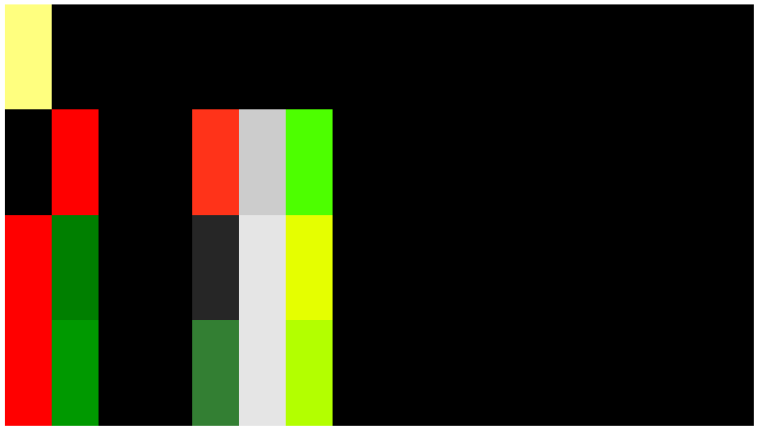
Kernel code:

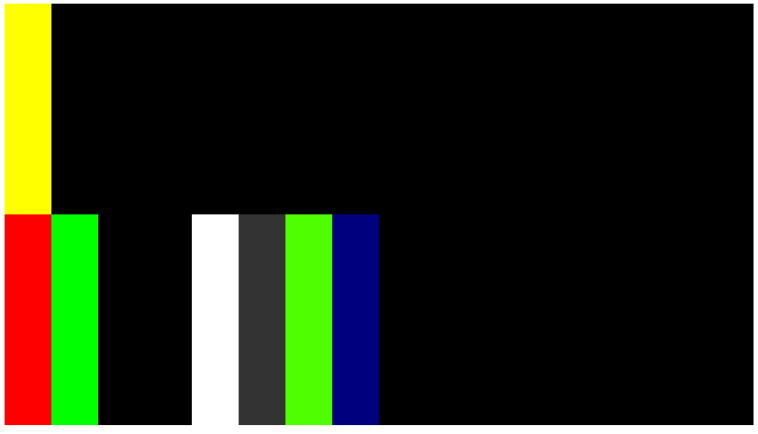
```
kernel void square(global int *dest, global int *data)
{
    int idx = get_global_id(0);
    dest[idx] = data[idx] * data[idx];
}
```

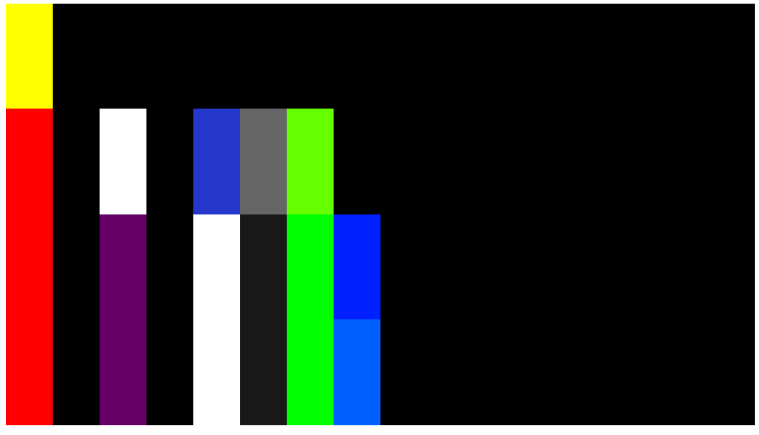
GPU Παράδειγμα σε GLSL

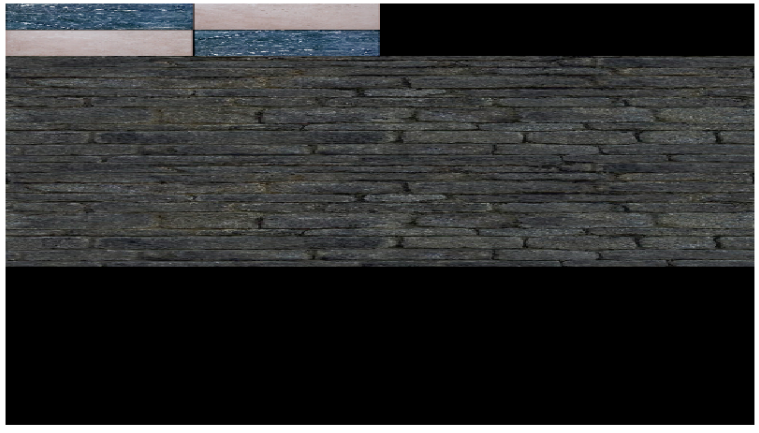
- Υλοποίηση σαν GLSL fragment shader (pixel shader)
- Κάθε "thread" εκτελείται για συντεταγμένες (`gl_FragCoord.x`, `gl_FragCoord.y`)
- Τα data structures της σκηνής κωδικοποιούνται σαν pixel rows σε texture images.
- Fullscreen OpenGL polygon (quad) γιά να εκτελεστεί ο shader για κάθε pixel.

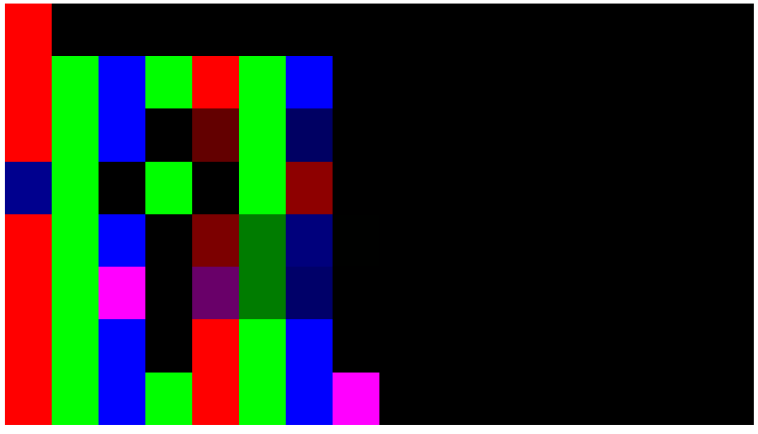
Scene data buffers











GPU ray tracer example video