**OpenCL Intro**

<https://www.youtube.com/watch?v=rpMNTTlTUok>

00:00 - Вступление

00:07 - Проверка совместимости видеокарты

00:29 - Загрузка и установка драйвера для видеокарты

01:29 - Установка библиотек OpenCL

02:01 - Подготовка проекта

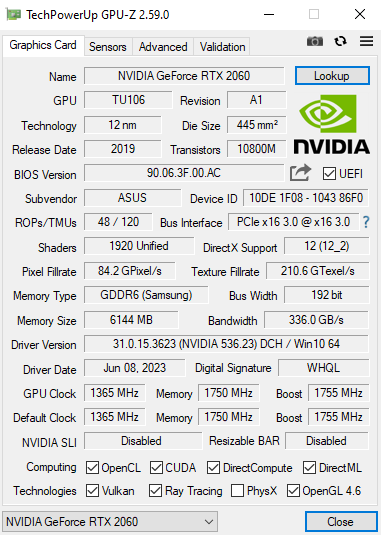
02:45 - Создание пакетного файла для компиляции программы

03:14 - Проверка подключения OpenCL

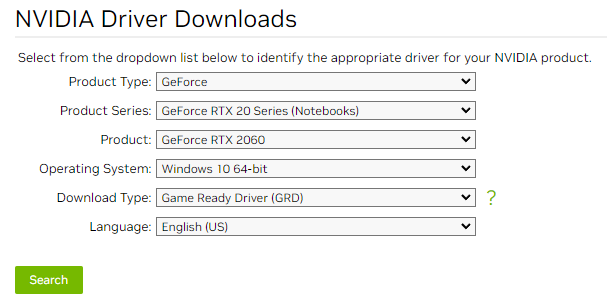
03:43 - Написание программы возведения чисел в квадрат

05:23 - Проверка работоспособности программы

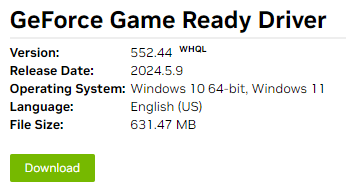
05:49 - Заключение



<https://www.nvidia.com/download/index.aspx>

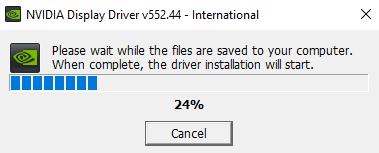


<https://www.nvidia.com/download/driverResults.aspx/224484/en-us/>

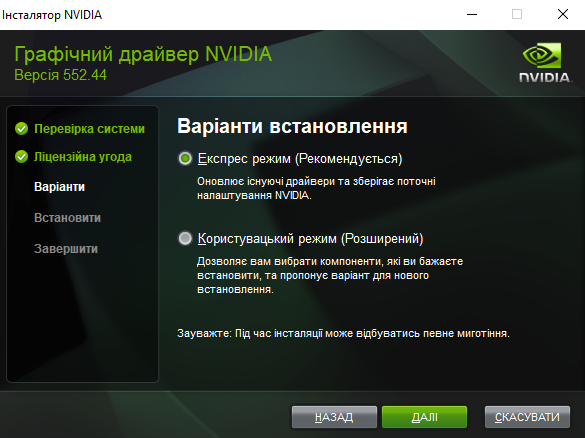


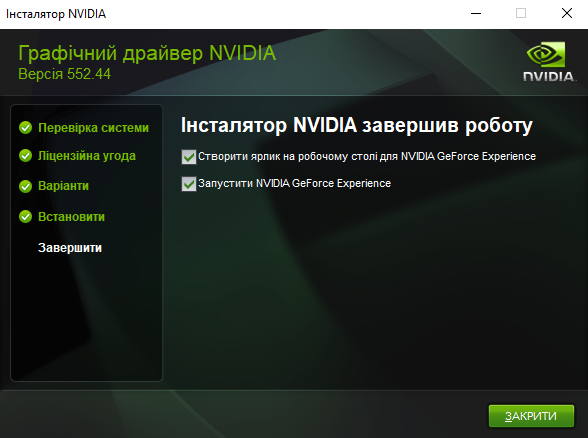
**Download** 552.44-notebook-win10-win11-64bit-international-dch-whql.exe

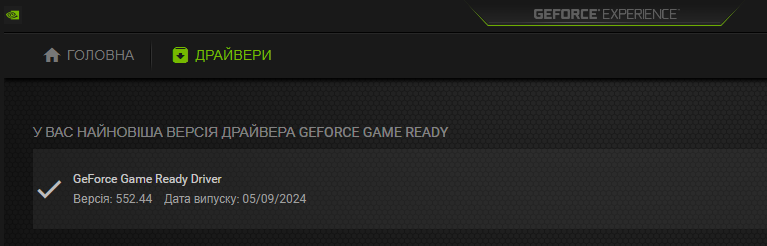
**Install to path** C:\NVIDIA\DisplayDriver\552.44\Win11\_Win10-DCH\_64\International







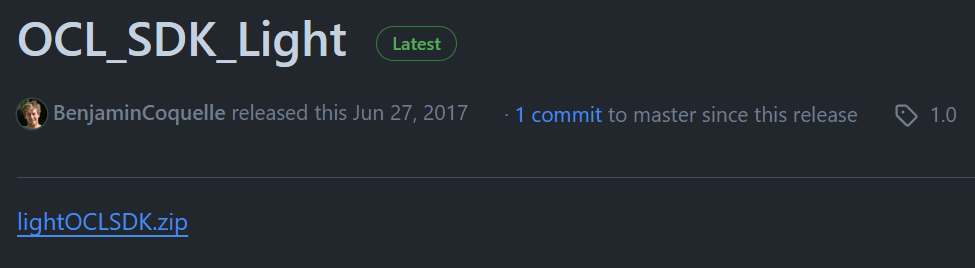




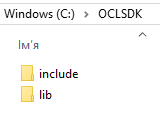
**Reboot PC**

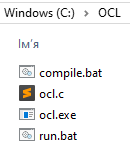
<https://github.com/GPUOpen-LibrariesAndSDKs/OCL-SDK>

<https://github.com/GPUOpen-LibrariesAndSDKs/OCL-SDK/releases/tag/1.0>



**Download** lightOCLSDK.zip





**compile.bat** *(*@gcc -o… *- disable out console text)*

gcc -o ocl.exe ocl.c -I "C:/OCLSDK/include/" -L "C:/OCLSDK/lib/x86\_64/" -lOpenCL

paese

**run.bat**

ocl.exe

paese

**ocl.c**

#include <CL/cl.h> // For APPLE <OpenCL/cl.h>

#include <stdio.h>

int **main**(void) {

cl\_int err;

cl\_uint numPlatforms;

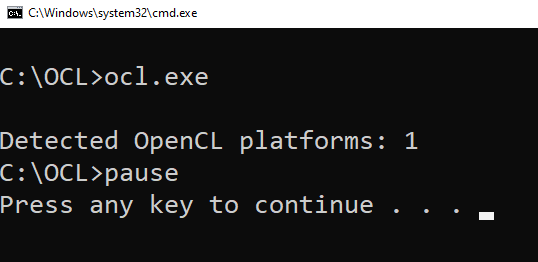
err = clGetPlatformIDs(0, NULL, &numPlatforms);

if (CL\_SUCCESS == err) {printf("\nDetected OpenCL platforms: %d", numPlatforms);}

else {printf("\nError calling clGetPlatformIDs. Error code: %d", err);}

return 0;

}



**ocl2.c**

#include <CL/cl.h> // For APPLE <OpenCL/cl.h>

#include <stdio.h>

#include <stdlib.h>

#define VECTOR\_SIZE 1024

// Программа выполняемая многопоточно

const char \*kernelSquare =

"\_\_kernel void **kernel\_square**(\_\_global float \*A, \_\_global float \*B) { \n"

" int index = get\_global\_id(0); // Получение номера потока \n"

" B[index] = A[index] \* A[index]; \n"

"} \n";

int **main**(void) {

int i;

// Инициализация векторов А и В

float \*A = (float \*)malloc(sizeof(float) \* VECTOR\_SIZE);

float \*B = (float \*)malloc(sizeof(float) \* VECTOR\_SIZE);

for (i = 0; i < VECTOR\_SIZE; i++) {A[i] = i; B[i] = 0;}

// Получение информации о доступных платформах

cl\_platform\_id \*platforms = NULL;

cl\_uint num\_platforms;

cl\_int clStatus = clGetPlatformIDs(0, NULL, &num\_platforms);

platforms = (cl\_platform\_id \*)malloc(sizeof(cl\_platform\_id) \* num\_platforms);

clStatus = clGetPlatformIDs(num\_platforms, platforms, NULL);

// Получение списка устройств и выбор устройства для исполнения кода

cl\_device\_id \*device\_list = NULL;

cl\_uint num\_devices;

clStatus = clGetDeviceIDs(platforms[0], CL\_DEVICE\_TYPE\_GPU, 0, NULL, &num\_devices);

device\_list = (cl\_device\_id \*)malloc(sizeof(cl\_device\_id) \* num\_devices);

clStatus = clGetDeviceIDs(platforms[0], CL\_DEVICE\_TYPE\_GPU, num\_devices, device\_list, NULL);

// Создание контекста для каждого устройства

cl\_context context;

context = clCreateContext(NULL, num\_devices, device\_list, NULL, NULL, &clStatus);

// Создание очереди команд (OpenCL < 2.0)

//cl\_command\_queue command\_queue = clCreateCommandQueue(context, device\_list[0], 0, &clStatus);

// Создание очереди команд (OpenCL >= 2.0)

cl\_command\_queue command\_queue = clCreateCommandQueueWithProperties(context, device\_list[0], 0, &clStatus);

// Создание буфера памяти для каждого вектора

cl\_mem A\_clmem = clCreateBuffer(context, CL\_MEM\_READ\_ONLY, VECTOR\_SIZE \* sizeof(float), NULL, &clStatus);

cl\_mem B\_clmem = clCreateBuffer(context, CL\_MEM\_WRITE\_ONLY, VECTOR\_SIZE \* sizeof(float), NULL, &clStatus);

// Скопировать буфер A на устройство

clStatus = clEnqueueWriteBuffer(command\_queue, A\_clmem, CL\_TRUE, 0, VECTOR\_SIZE \* sizeof(float), A, 0, NULL, NULL);

// Создать программу kernelSource

cl\_program program = clCreateProgramWithSource(context, 1, (const char \*\*)&kernelSquare, NULL, &clStatus);

// Собрать программу

clStatus = clBuildProgram(program, 1, device\_list, NULL, NULL, NULL);

// Создать процесс на устройстве

cl\_kernel kernel = clCreateKernel(program, "kernel\_square", &clStatus);

// Передать аргументы в программу

clStatus = clSetKernelArg(kernel, 0, sizeof(cl\_mem), (void \*)&A\_clmem);

clStatus = clSetKernelArg(kernel, 1, sizeof(cl\_mem), (void \*)&B\_clmem);

// Выполнить программу

size\_t global\_size = VECTOR\_SIZE;

size\_t local\_size = 64;

clStatus = clEnqueueNDRangeKernel(command\_queue, kernel, 1, NULL, &global\_size, &local\_size, 0, NULL, NULL);

// Скопировать буфер B на хостовое устройство

clStatus = clEnqueueReadBuffer(command\_queue, B\_clmem, CL\_TRUE, 0, VECTOR\_SIZE \* sizeof(float), B, 0 , NULL, NULL);

// Ожидание завершения всех команд

clStatus = clFlush(command\_queue);

clStatus = clFinish(command\_queue);

// Вивод результатов

for (i = 0; i < VECTOR\_SIZE; i++) {printf("%f \* %f = %f\n", A[i], A[i], B[i]);}

// Освобождение памяти

clStatus = clReleaseKernel(kernel);

clStatus = clReleaseProgram(program);

clStatus = clReleaseMemObject(A\_clmem);

clStatus = clReleaseMemObject(B\_clmem);

clStatus = clReleaseCommandQueue(command\_queue);

clStatus = clReleaseContext(context);

free(A);

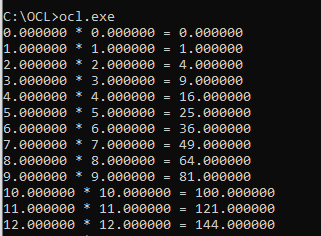
free(B);

free(platforms);

free(device\_list);

return 0;

}



**Source code:** <https://gist.github.com/yohanesgultom/b7e32f7649ac39e00ad65bcb83dfd72e>

// Программа выполняемая многопоточно

const char \*kernelSquare =

"\_\_kernel void **kernel\_square**(\_\_global float \*A, \_\_global float \*B) { \n"

" int index = get\_global\_id(0); // Получение номера потока \n"

" B[index] = A[index] \* A[index]; \n"

"} \n";

**Способ создания строковой переменной kernelSquare из файла**

// Tested OpenCL on: CL\_PLATFORM\_VERSION: OpenCL 1.2 CUDA 9.0.282

#define CL\_USE\_DEPRECATED\_OPENCL\_1\_2\_APIS

// Create a program from the kernel source

#define MAX\_SOURCE\_SIZE (0x100000)

char \***oclLoadProgSource**(char \*fileName, size\_t \*source\_size) {

// Load the source code containing the kernel

FILE \*fp = fopen(fileName, "r");

if (!fp) {fprintf(stderr, "Failed to load kernel.\n"); exit(1);}

char \*source\_str = (char\*)malloc(MAX\_SOURCE\_SIZE);

\*source\_size = fread(source\_str, 1, MAX\_SOURCE\_SIZE, fp);

fclose(fp);

return source\_str;

}

char \*kernel\_filename = "OpenclTextProg.cl";

size\_t kernelLength;

char \*kernelSquare = NULL;

kernelSquare = **oclLoadProgSource**(kernel\_filename, &kernelLength);

// printf("%s\n", kernel\_content);