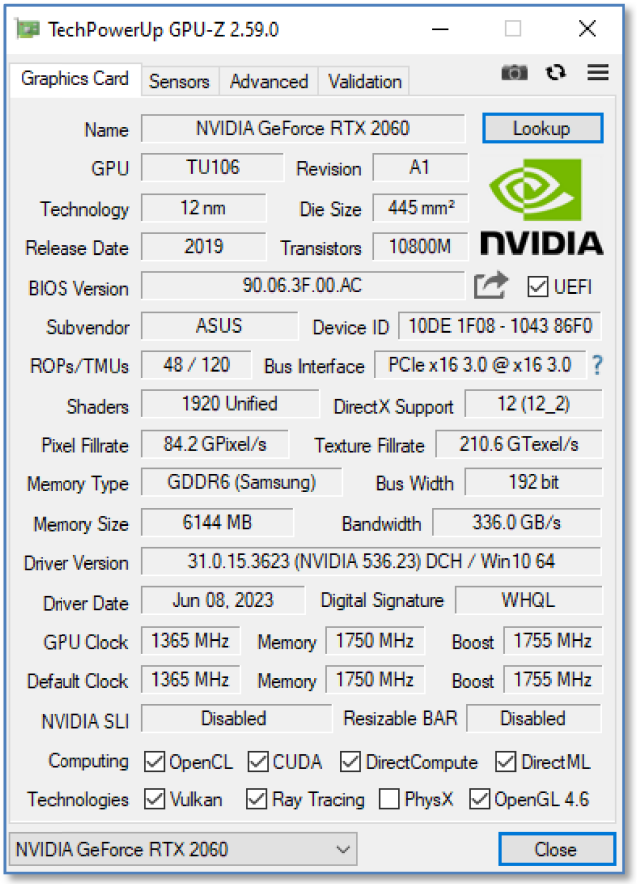
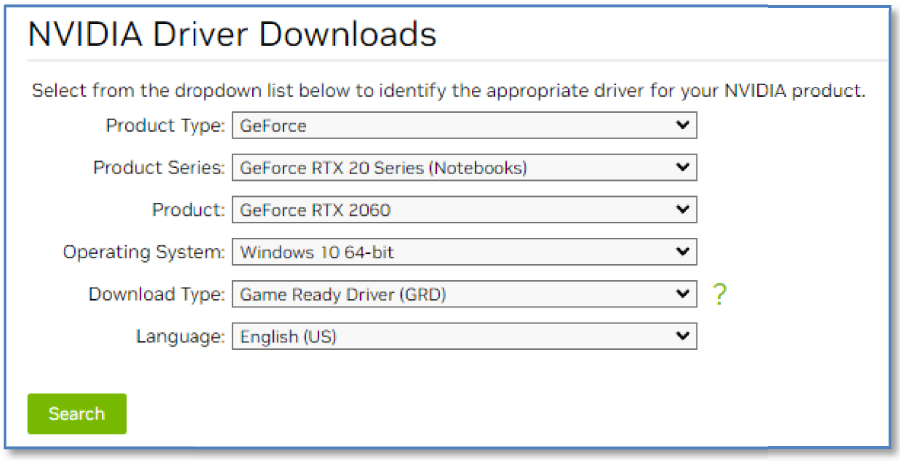
OpenCL Intro

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

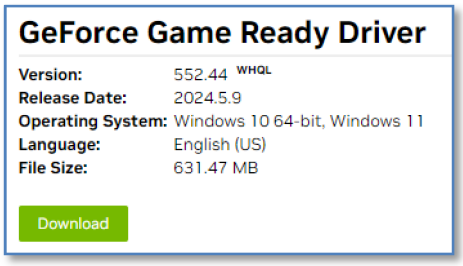
00:00 - Вступление
00:07 - Проверка совместимости видеокарты
00:29 - Загрузка и установка драйвера для видеокарты
01:29 - Установка библиотек OpenCL
02:01 - Подготовка проекта
02:45 - Создание пакетного файла для компиляции программы
03:14 - Проверка подключения OpenCL
03:43 - Написание программы возведения чисел в квадрат
05:23 - Проверка работоспособности программы
05:49 - Заключение



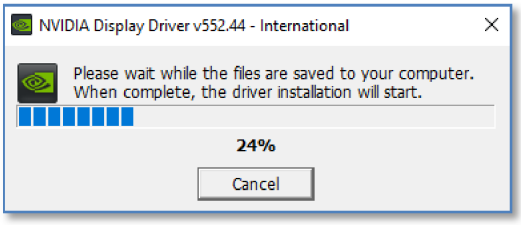
https://www.nvidia.com/download/index.aspx

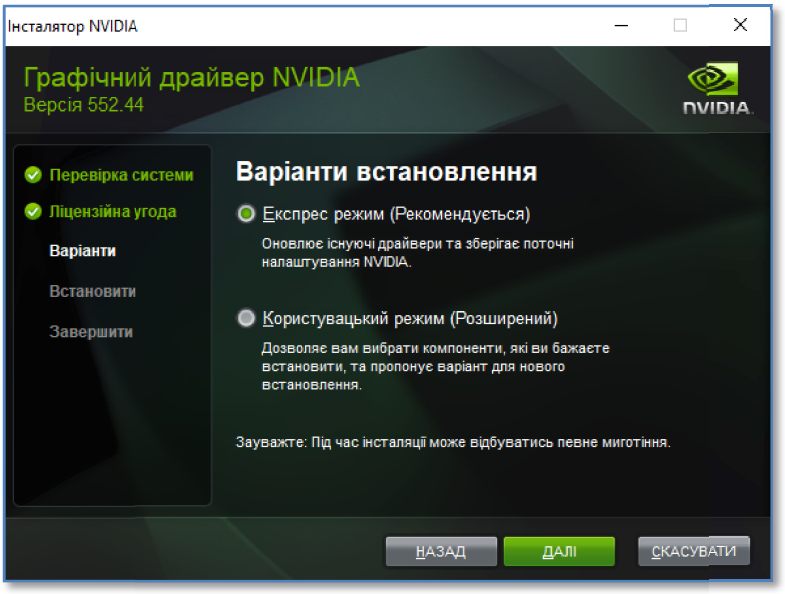


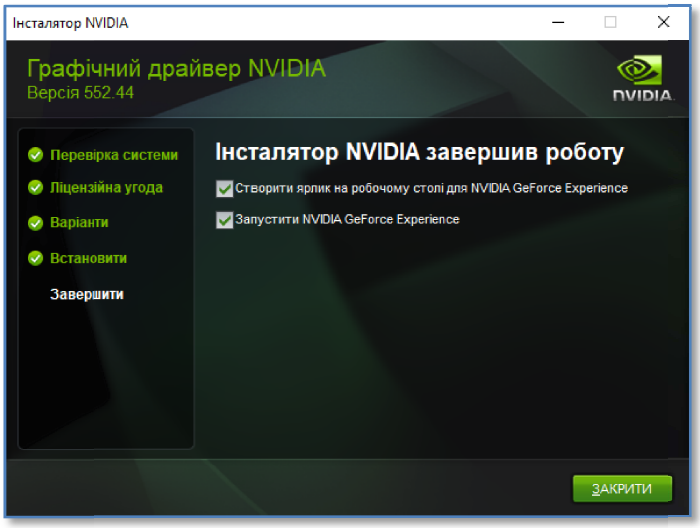
https://www.nvidia.com/download/driverResults.aspx/224484/en-us/

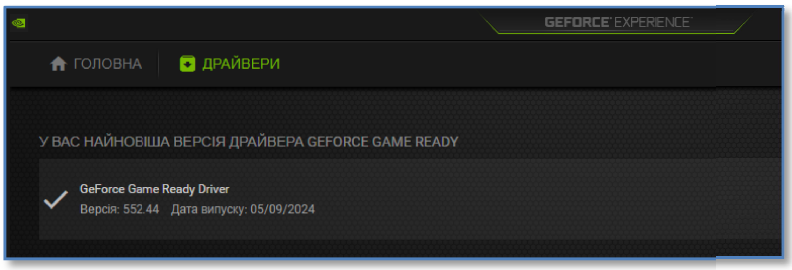


Download 552.44-notebook-win10-win11-64bit-international-dch-whql.exe
Install to path C:\NVIDIA\DisplayDriver\552.44\Win11_Win10-DCH_64\International



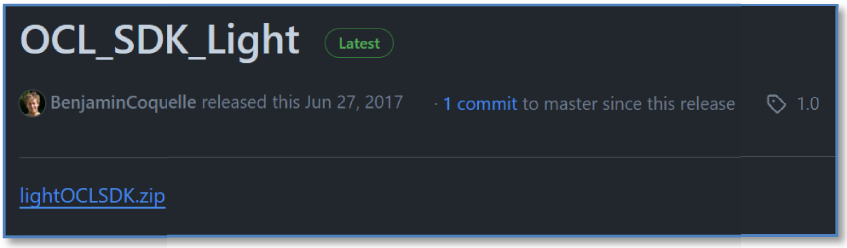




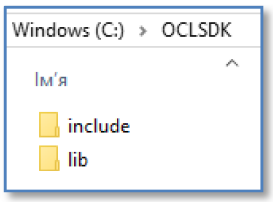


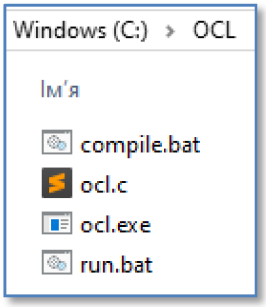
Reboot PC

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
```

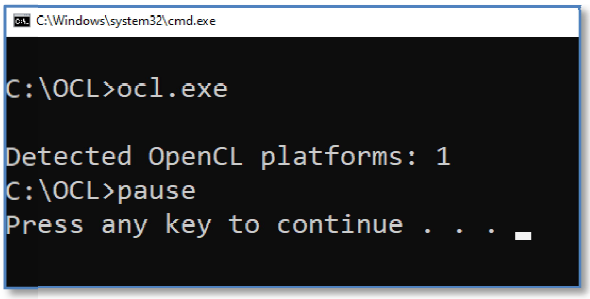
run.bat

```
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 A и B
        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;
}
```

```
C:\OCL>ocl.exe
0.000000 * 0.000000 = 0.000000
1.000000 * 1.000000 = 1.000000
2.000000 * 2.000000 = 4.000000
3.000000 * 3.000000 = 9.000000
4.000000 * 4.000000 = 16.000000
5.000000 * 5.000000 = 25.000000
6.000000 * 6.000000 = 36.000000
7.000000 * 7.000000 = 49.000000
8.000000 * 8.000000 = 64.000000
9.000000 * 9.000000 = 81.000000
10.000000 * 10.000000 = 100.000000
11.000000 * 11.000000 = 121.000000
12.000000 * 12.000000 = 144.000000
```

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);
```