Hello World

KOREA UNIVERSITY

KUCG

In this lecture, you will learn

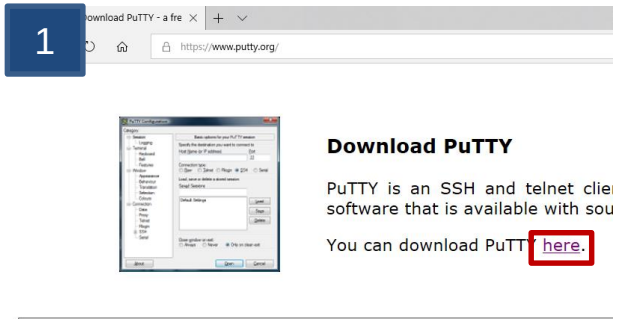
- **GPU Server**
  - Remind Connecting Server
  - Transfer file to Server
  - “Hello World!” with GPU Server
- **NVIDIA’s CUDA**
  - Architecture
  - Language Syntax
  - CUDA Compiler
  - “Hello World!” with GPU
- **OpenGL**
  - How to make OpenGL Window?
  - Simple Drawing with OpenGL
  - Compile on the Server

GPU Server

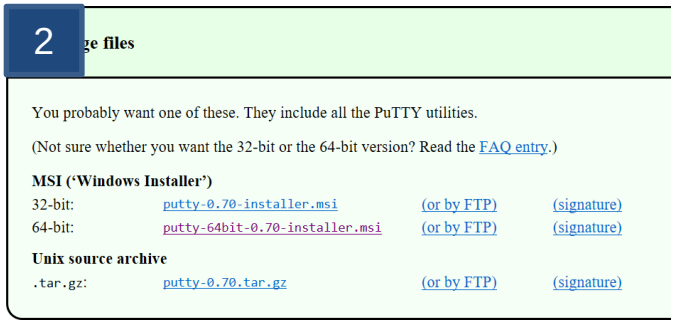
Remind Connecting Server with Putty

- PuTTY를 다운로드 https://www.putty.org/

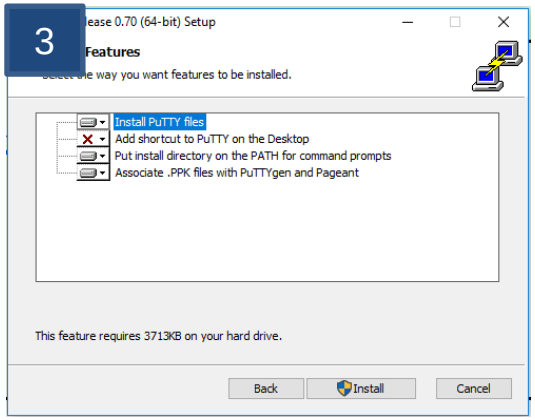
1



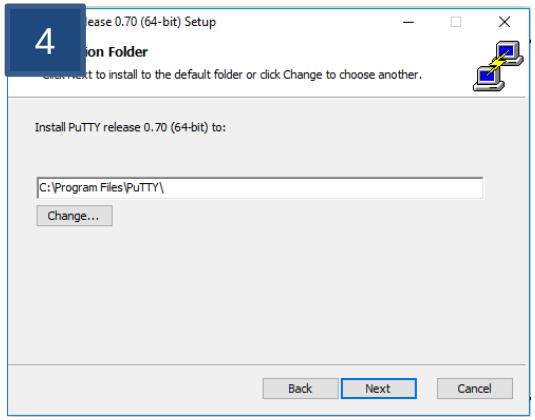
2



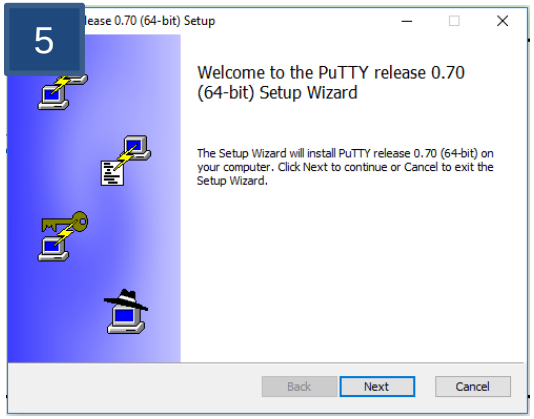
3



4

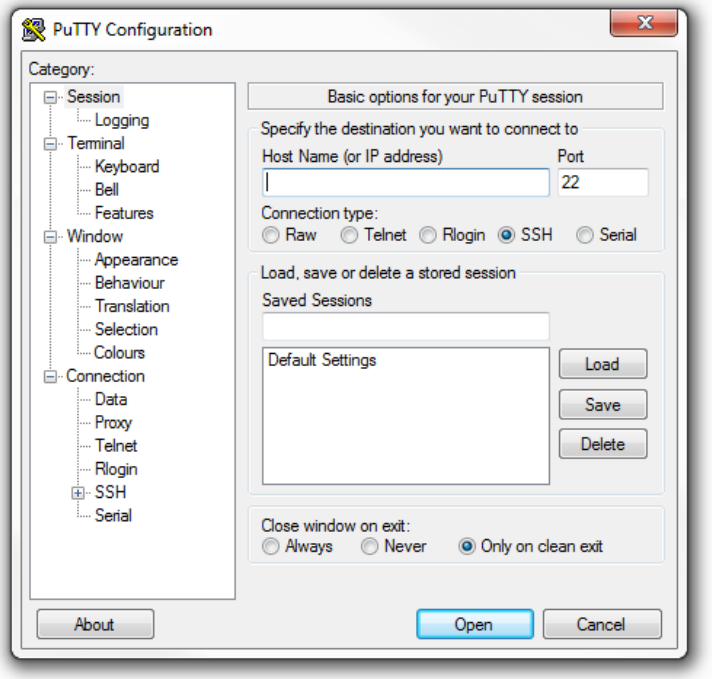


5



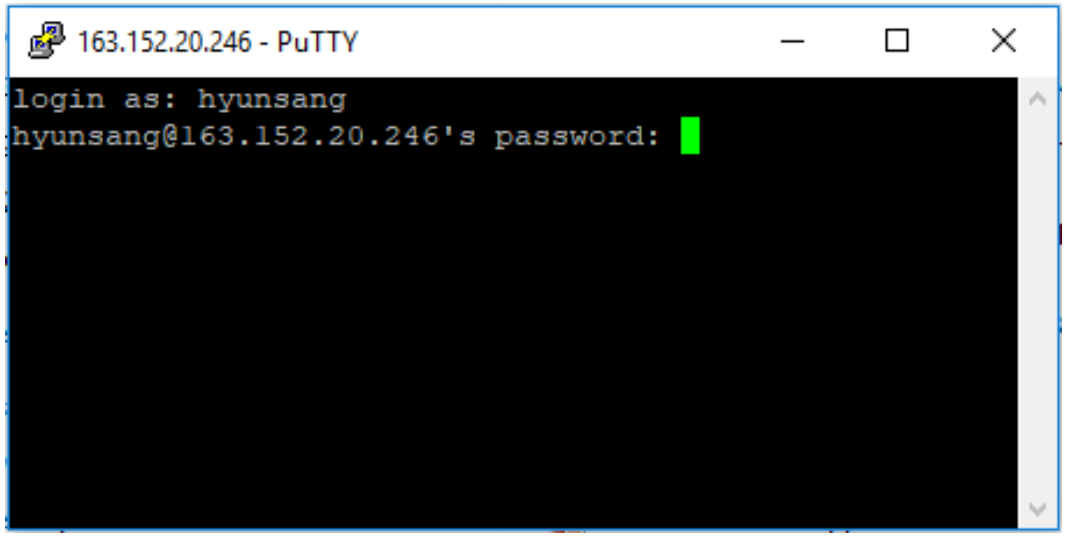
Remind Connecting Server with Putty

- 실행 후, Host Name에 서버 ip (163.152.20.246)를 입력한 후 Open



Remind Connecting Server with Putty

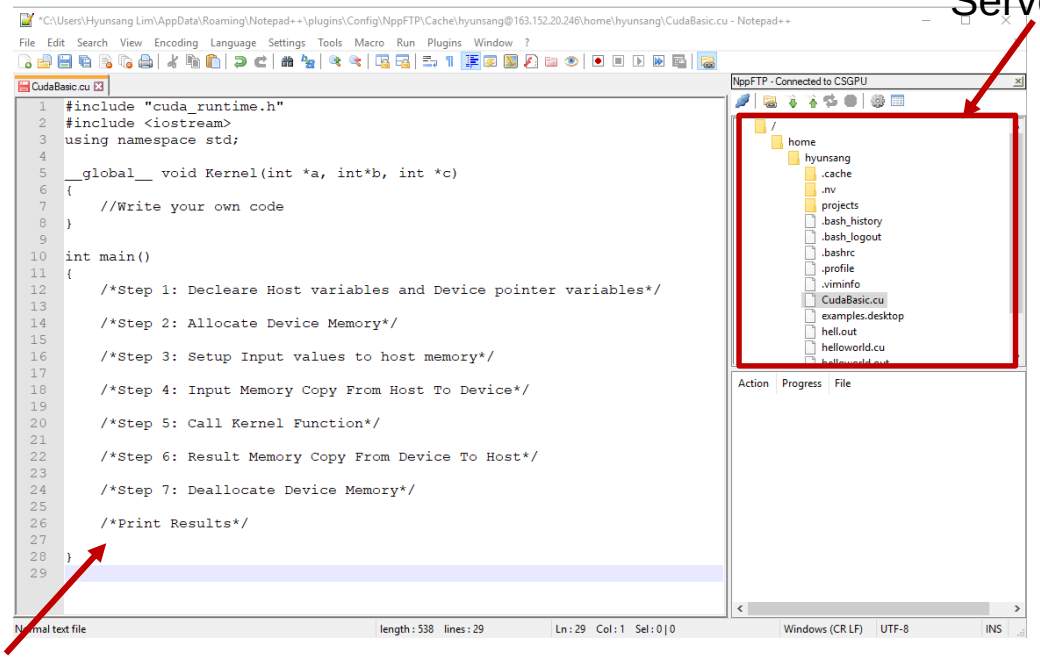
- 할당된 id와 password를 이용하여 로그인



/home/UserID/ 디렉토리에서 작업수행.

Sending Source Code to Server

- **Example using nppFTP.**



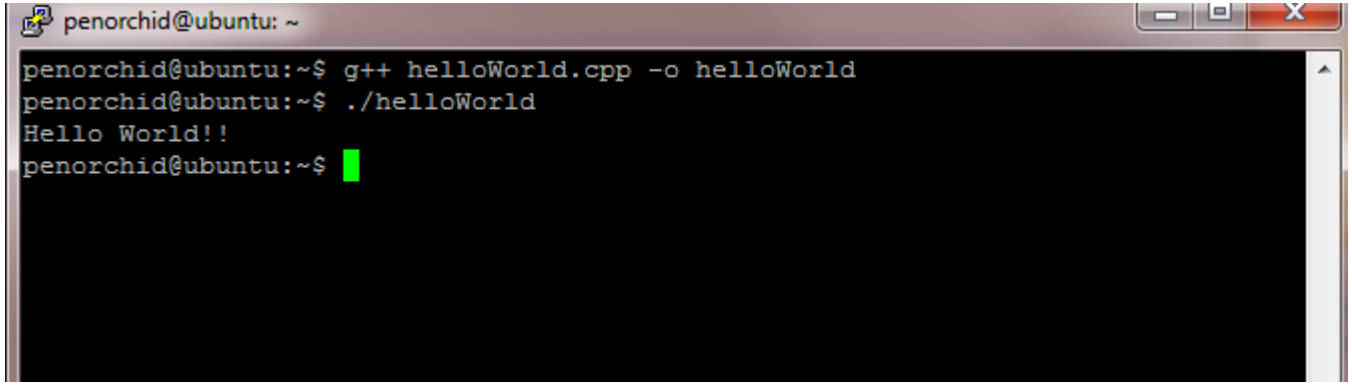
By saving the code, the code is transferred to the server

Hello World on GPU Server

- **Write Code and Save**

```
#include <stdio.h>

int main(void){
    printf("Hello World!!\n");
    return 0;
}
```

- **Compile and print result**



Compiling using Makefile : CUDA

- Makefile makes a compile procedure automatically.
- Create file named GNUmakefile, Makefile, makefile
- One file per folder is possible (Making folder: mkdir foldername)

vim Makefile

-Makefile code

Macro declaration

```
CC = nvcc
TARGET = cudaAdd_exe
```

//Compiler name

//Name of Output run file

Main order

```
$(TARGET) :
	$(CC) CudaAdd.cu -o $(TARGET)
```

Name of the source code

- Just Type 'make', then compile and update of the libraries would be done automatically.

Compiling using Makefile : OpenGL

- Makefile makes a compile procedure automatically.
- Create file named GNUmakefile, Makefile, makefile
- One file per folder is possible (Making folder: mkdir foldername)

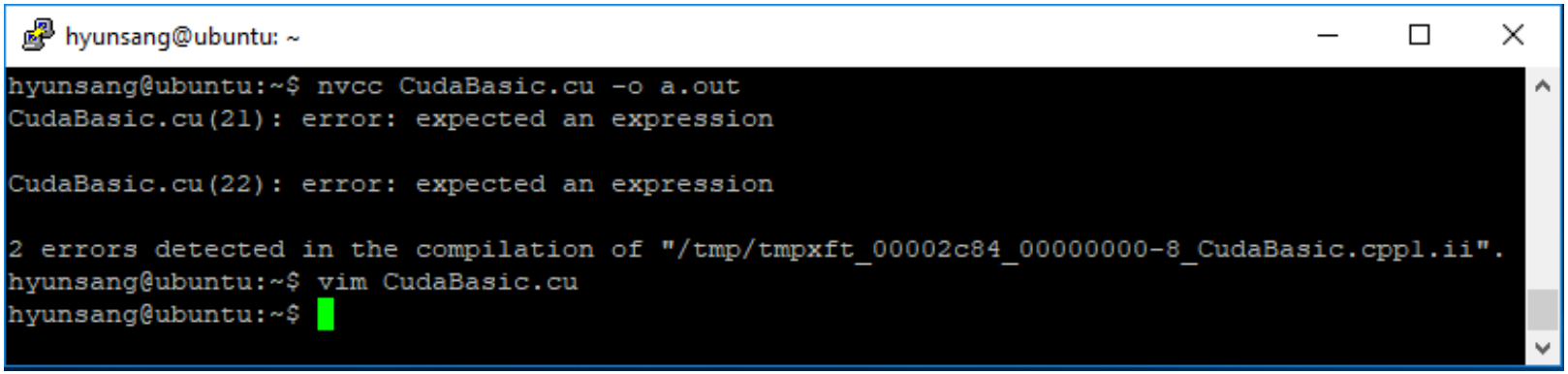
vim Makefile

```
PP = g++
TARGET = openGLBasic_exe
$(TARGET):
	$(PP) openGLBasic.cpp -o $(TARGET)  -lX11 -lGL -lglut
```

- Just Type ‘make’, then compile and update of the libraries would be done automatically.

Checking Error Message

- If a compile error occurs, error message is displayed.
- Modify the code using notepad++



- i 키를 눌러 Insert Mode로 진입
- 소스 코드 디버깅
- ESC 키를 눌러 Command Mode로 진입
- :wq를 입력하여 파일을 저장하고 나가기

```
#include<iostream>

using namespace std;

int main()
{
        cout<<"hello world!"<<endl;
        return 0;
}
```

Procedure Summary

(1) Connecting server with PuTTY.

(2) Sending Source code to Server.

(3) Compiling with Makefile.

(4) Checking error and Debug.

CUDA

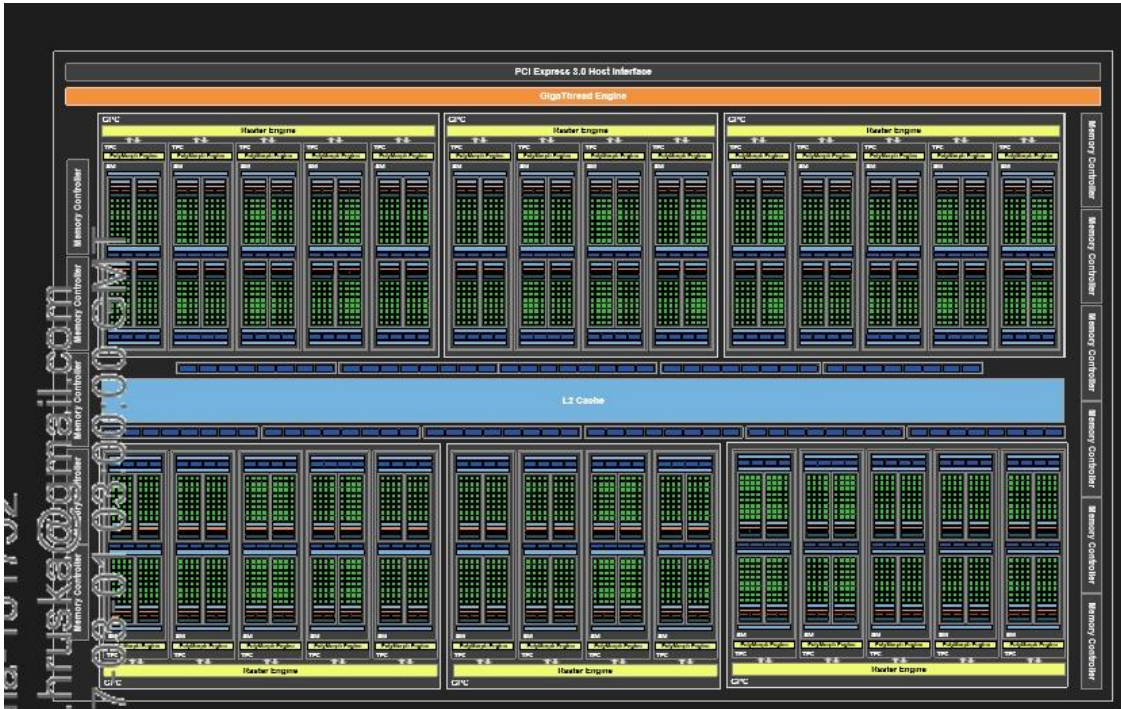
CUDA:
Compute Unified Device Architecture

- **A general purpose programming model on NVIDIA GPUs.**

- **Enables GPU memory access**

- **GPU as a computation device that:**
  - is a co-processor to CPU
  - has its own memory
  - runs many threads in parallel

GPU Spec@GPU Server



12B Transistors

1.6 GHz Boost, 2 GHz OC

28 SMs, 128 cores each

3584 CUDA cores

28 Geometry units

224 Texture units

6 GPCs

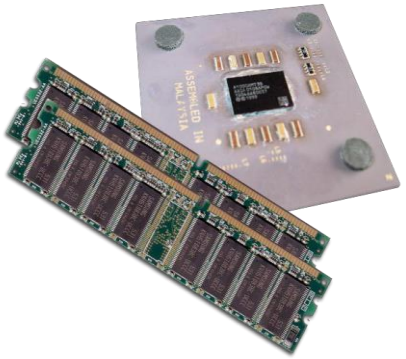
88 ROP units

352 bit GDDR5x

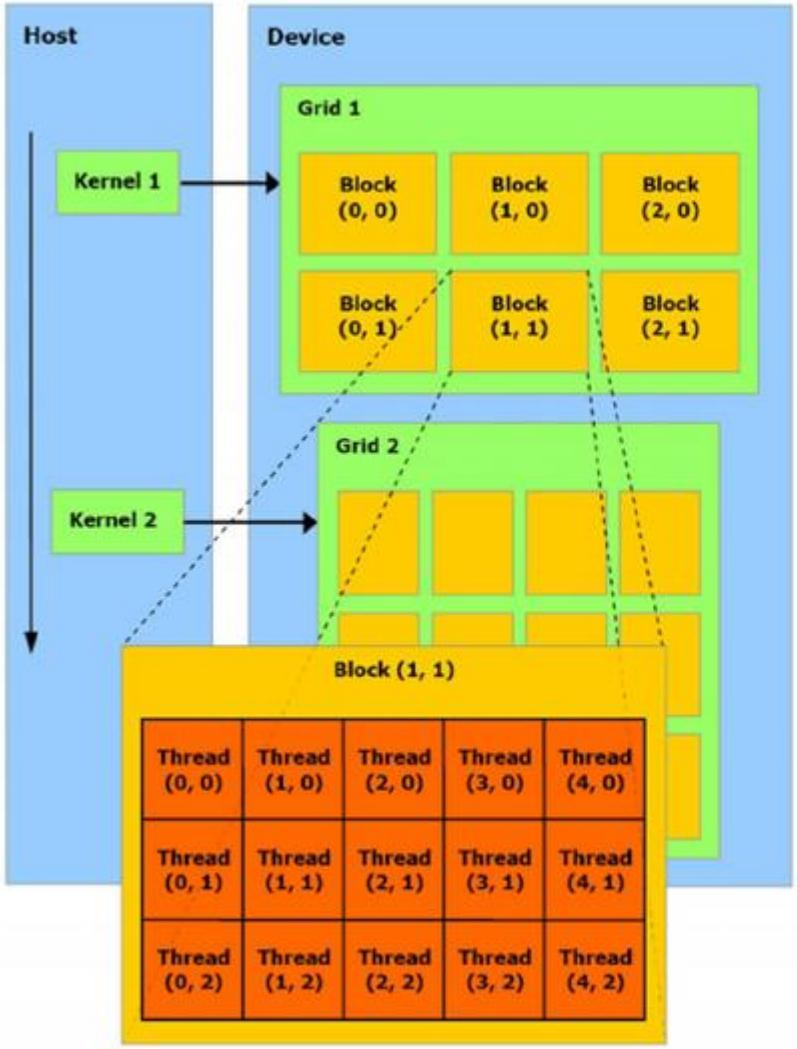
Memory Management

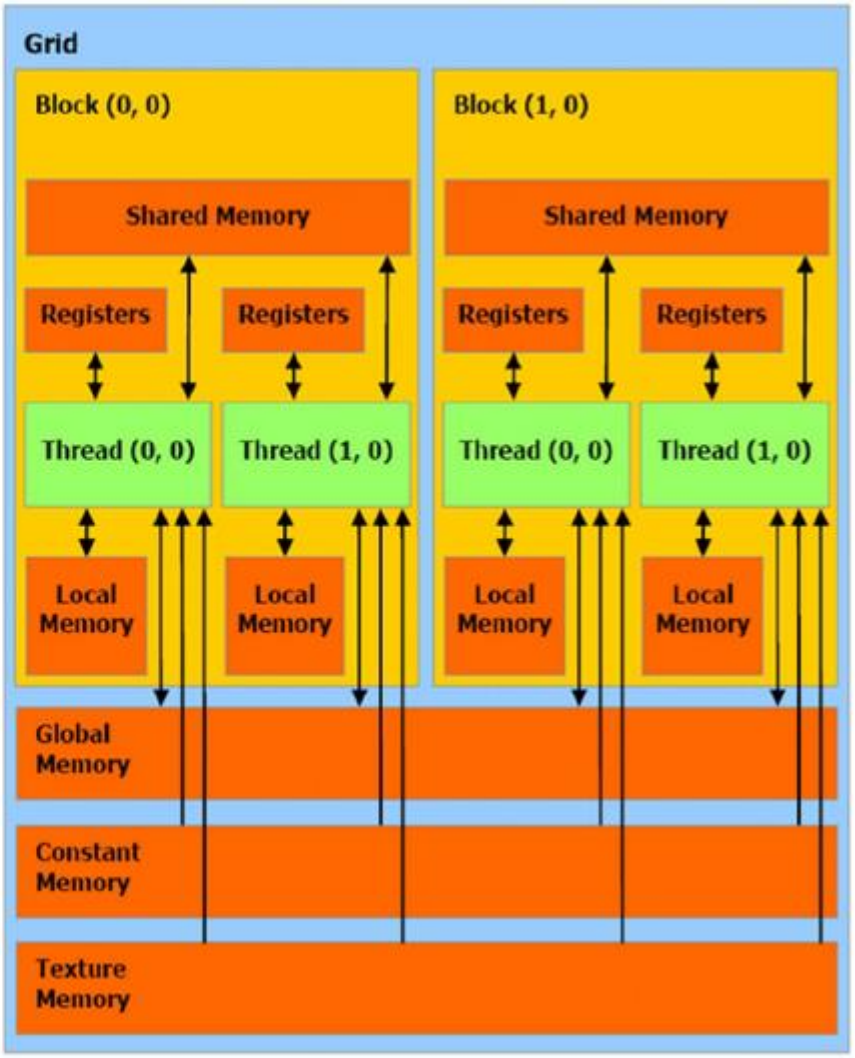
- **Host and device memory are separate entities**
  - *Device* pointers only point to GPU memory
    Can be passed to/from host code
    Cannot be dereferenced in host code

  - *Host* pointers only point to CPU memory
    Can be passed to/from device code
    Cannot be dereferenced in device code



CUDA Architecture

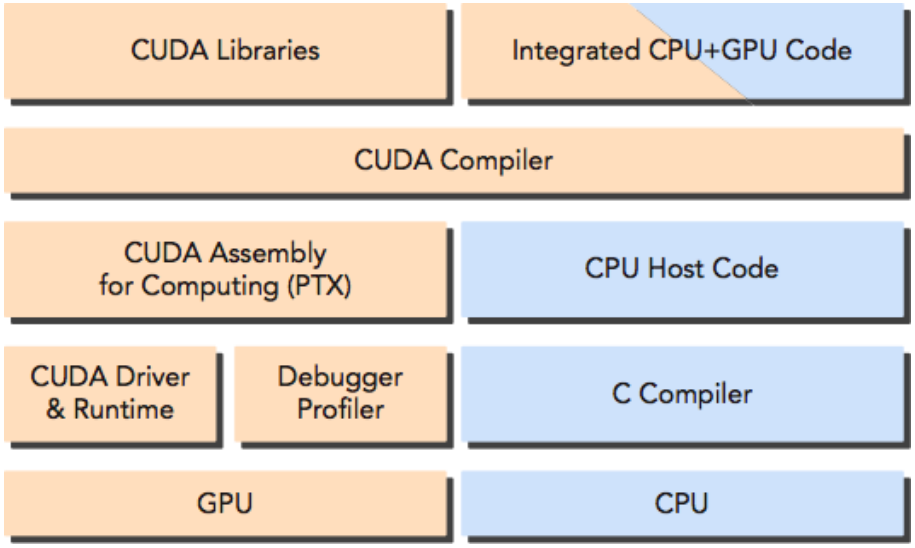




A CUDA program

- **A CUDA program consists of a mixture of the following two parts:**
  - Host code runs on CPU.
  - Device code runs on GPU.

- **NVIDIA's `nvcc` compiler separates the device code from the host code during the compilation process.**



CUDA Step:0
“Hello World!”

Hello World!

```
int main(void) {
        printf("Hello World!\n");
        return 0;
}
```

- Standard C program runs on the host
- NVIDIA compiler (nvcc) can compile same programs without *device* code

Hello World! with Device Code

```
__global__ void myKernel(void)
{
    //task on device
}

int main()
{
    cudaError_t cudaStatus = cudaSetDevice(0);
    if (cudaStatus != cudaSuccess) {
        cout << "cudaSetDevice failed!  Do you have a CUDA-capable GPU installed?" << endl;
    }

    myKernel << <1, 1 >> > ();
    printf("Hello World!\n");
    return 0;
}
```

- Two new syntactic elements…
  - __global__
  - <<<1,1>>>

Kernel Function

```
__global__ void mykernel(void) {
}
```

- CUDA C/C++ keyword `__global__` indicates a function that:
  - Runs on the device
  - Is called from host code

- `nvcc` separates source code into host and device components
  - Device functions (e.g. `mykernel()`) processed by NVIDIA compiler
  - Host functions (e.g. `main()`) processed by standard host compiler
    - `gcc`, `g++`, `etc.`

Kernel call syntax

```
mykernel<<<1,1>>>();
```

- Triple angle brackets mark a call from *host* code to *device* code
  - Also called a “kernel launch”
- That’s all that is required to execute a function on the GPU!

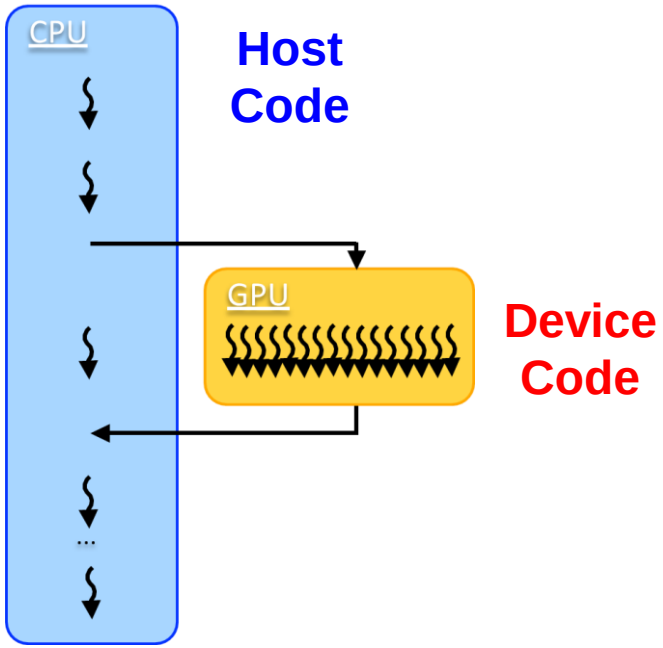
GPU Device Initialize & Check

```
cudaError_t cudaStatus = cudaSetDevice(0);
if (cudaStatus != cudaSuccess) {
    cout << "cudaSetDevice failed!  Do you have a CUDA-capable GPU installed?" << endl;
}
```

- Checking whether proper GPU is mounted or not

Main & Device Code

```
__global__ void mykernel(void) {
}

int main(void) {
        mykernel<<<1,1>>>();
        printf("Hello World!\n");
        return 0;
}
```



- We should fill `mykernel()`
- Let's try adding two integers

CUDA Step:1
Computing On Device

Typical CUDA Programming Flow

1. Declare a tuple of Host & Device pointer variables
2. Allocate Device memory for the variables
3. Copy memory for Input values from Host to Device
4. Call Kernel Function
5. Copy memory for Result from Device to Host
6. Release Device memory
7. Print / Visualize Results

Declare a tuple of Host & Device pointer variables

- **Host Memory must be Pointer or Array variable**
  - Pointer variable

    ```
    int *a=(int*)malloc(sizeof(int)*3);
    int* b=(int*)malloc(sizeof(int)*3);
    int* c=(int*)malloc(sizeof(int)*3);
    ```

  - Array variable

    ```
    const int a[3] = { 1, 2, 3};const int b[3] = {10,20,30};    int c[3] = { 0 };
    ```

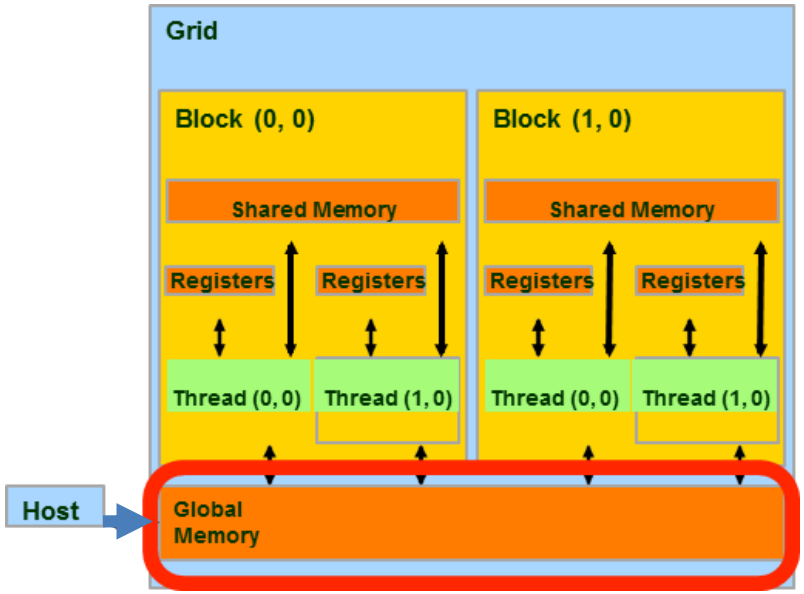
- **Device Memory must be Pointer variable**
  - Pointer variable

    ```
    int* dev_a; int* dev_b; int* dev_c;
    ```

Allocate Device Memory for the variable

- **Allocating memory in the Global Memory of Device and return pointer to it.**

```
int *dev_a;
cudaStatus = cudaMalloc((void**)&dev_a, N*sizeof(int));
if(cudaStatus != cudaSuccess){
    fprintf("cudaMalloc failed!");
}
```

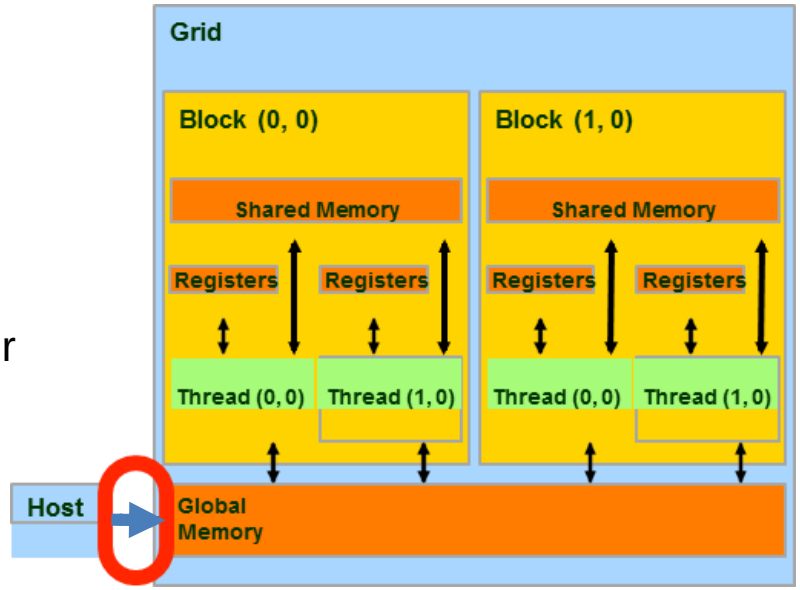


Copy memory for Input values from Host to Device

Destination Source

```
cudaStatus = cudaMemcpy(dev_a, a, size * sizeof(int), cudaMemcpyHostToDevice);
if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMemcpy failed!");
}
```

- `d_a` is pointers to device data
- `h_a` is pointers to host data
- `size` is the size of data in byte
- The keywords `cudaMemcpyHostToDevice` tell `cudaMemcpy()` the direction of data transfer



Call Kernel function

- Kernel call order in Host

```
addKernel <<<1, 1>>> (dev_c, dev_a, dev_b);
```

- Kernel function(Device code)

```
__global__ void add(int *a, int *b, int *c) {
        *c = *a + *b;
}
```

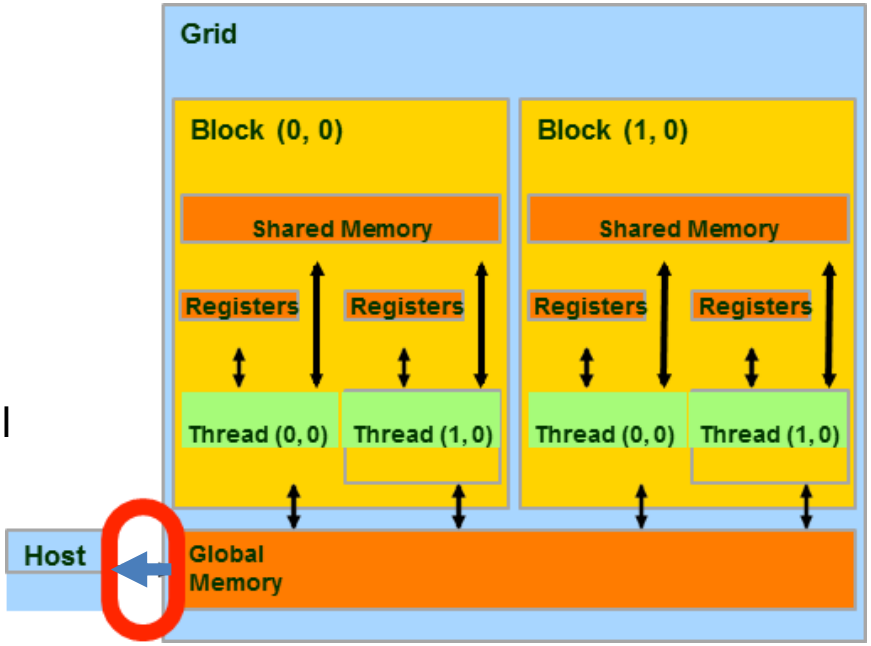
- The CUDA keyword `__global__` means:
  - `add()` is executed on the Device
  - `add()` is called only by the Host
  - Return type is only `void`

Copy memory for Result from Device to Host

Destination Source

```
cudaStatus = cudaMemcpy(c, dev_c, size * sizeof(int), cudaMemcpyDeviceToHost);
if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMemcpy failed!");
}
```

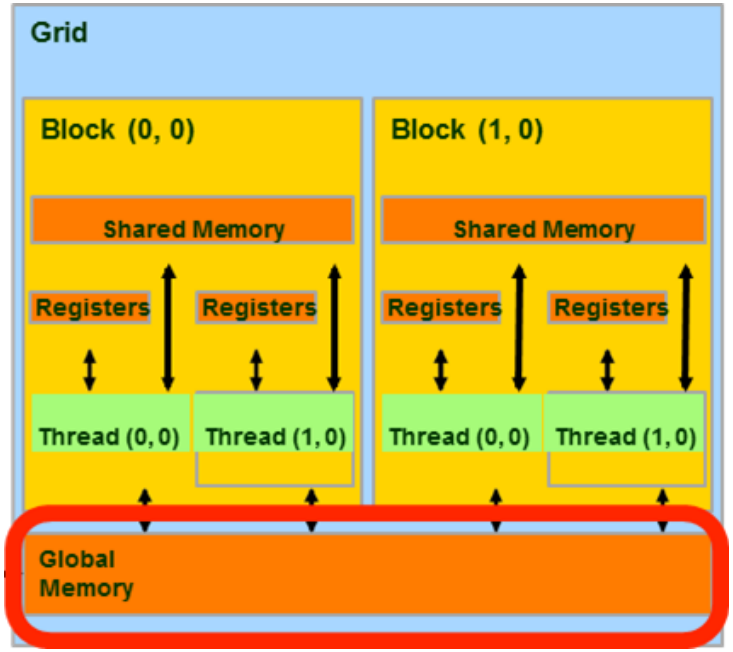
- `d_a` and `d_c` are pointers to device data
- `h_a` and `h_c` are pointers to host data
- `size` is the size of data in byte
- The keywords `cudaMemcpyDeviceToHost` tell `cudaMemcpy()` the direction of data transfer



Release the Device Memory

- **Releasing Memory: free object from device global memory**

```
cudaFree(dev_a);
cudaFree(dev_b);
cudaFree(dev_c);
```



CUDA Manual

- **CUDA NVIDIA Official Manual.**
  - Webpage

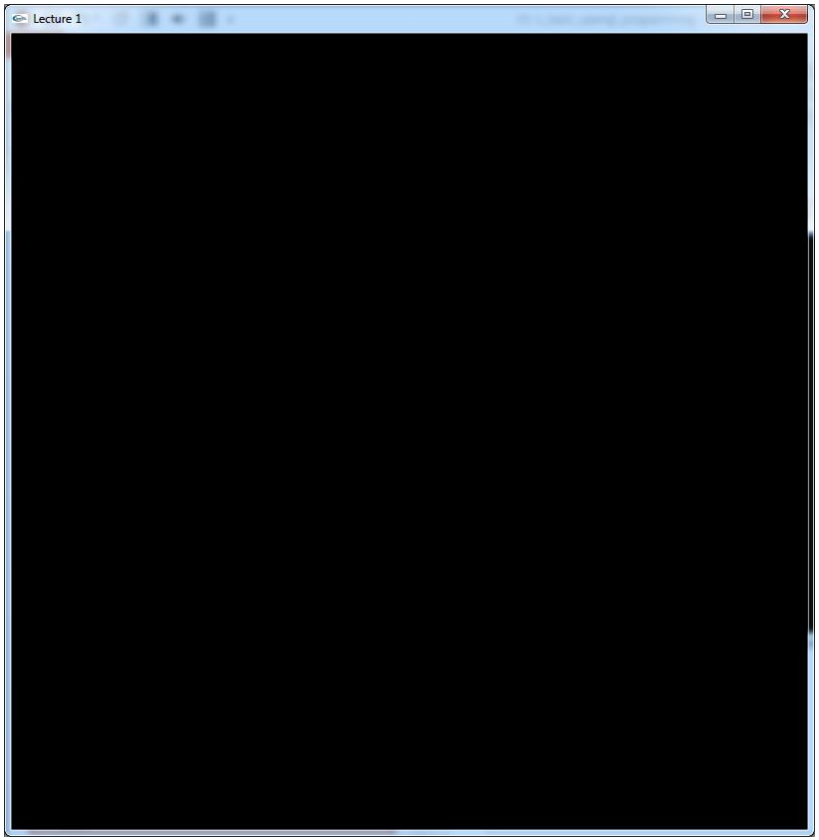
    https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#thread-hierarchy
  - PDF File (Version 4.2)

    https://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf

OpenGL: Hello World

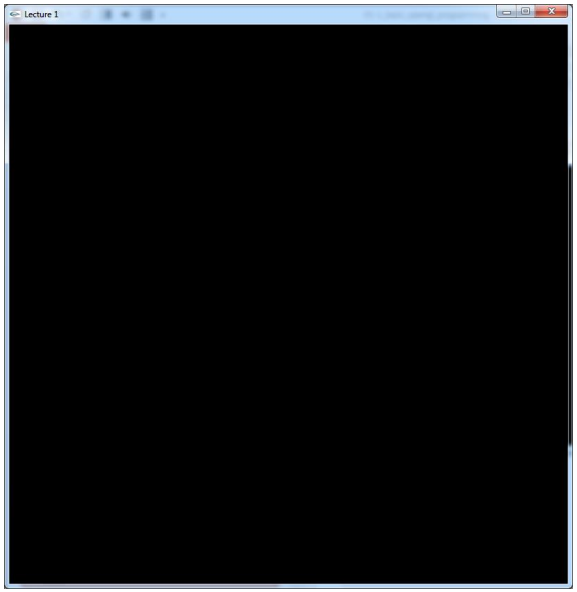
Creating an Empty Window

- OpenGL Window 만들기



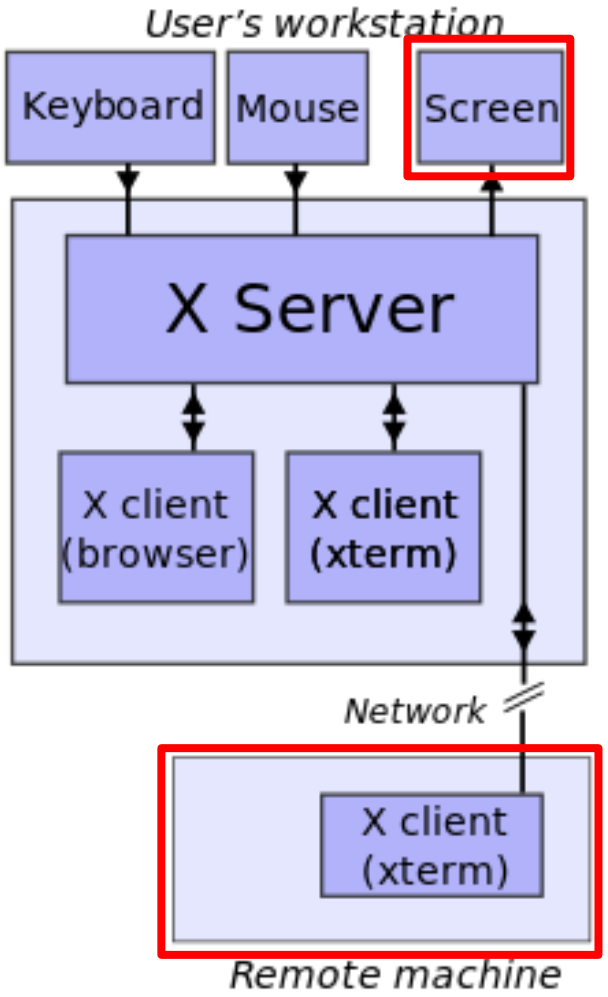
Sample Code: Empty Window on Windows

```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glu.h>
#include <gl/glut.h>

void display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    glutSwapBuffers();
}

int main(int argc, char **argv){
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);
    glutCreateWindow("Lecture");

    glutDisplayFunc(display);
    glutMainLoop();

    return 0;
}
```



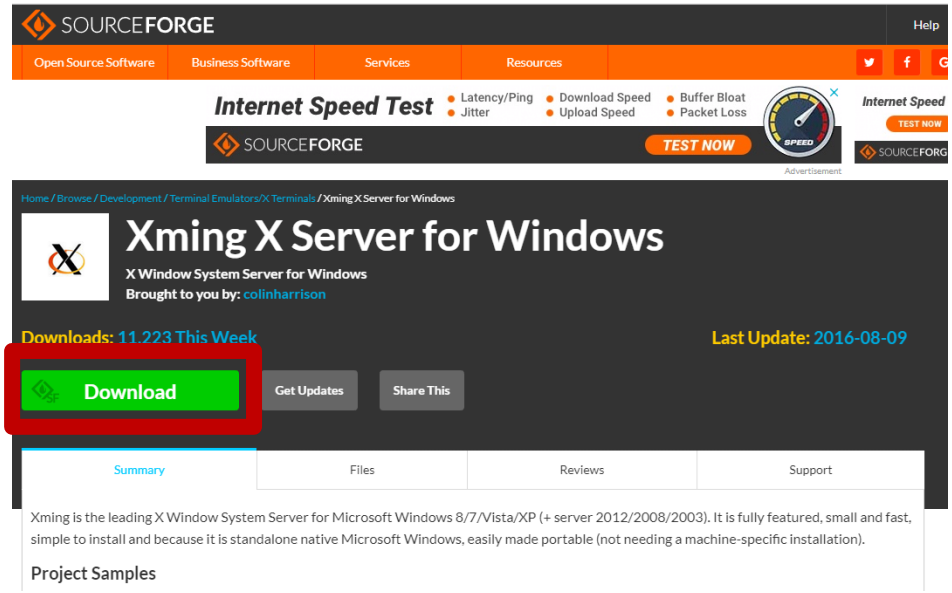
How to Create Window @ GPU Server

- **X Window System**
  - Windowing system for graphic displays
  - Common on UNIX-like computer operating systems.
  - An architecture-independent system for remote graphical user interfaces



X Server for Windows

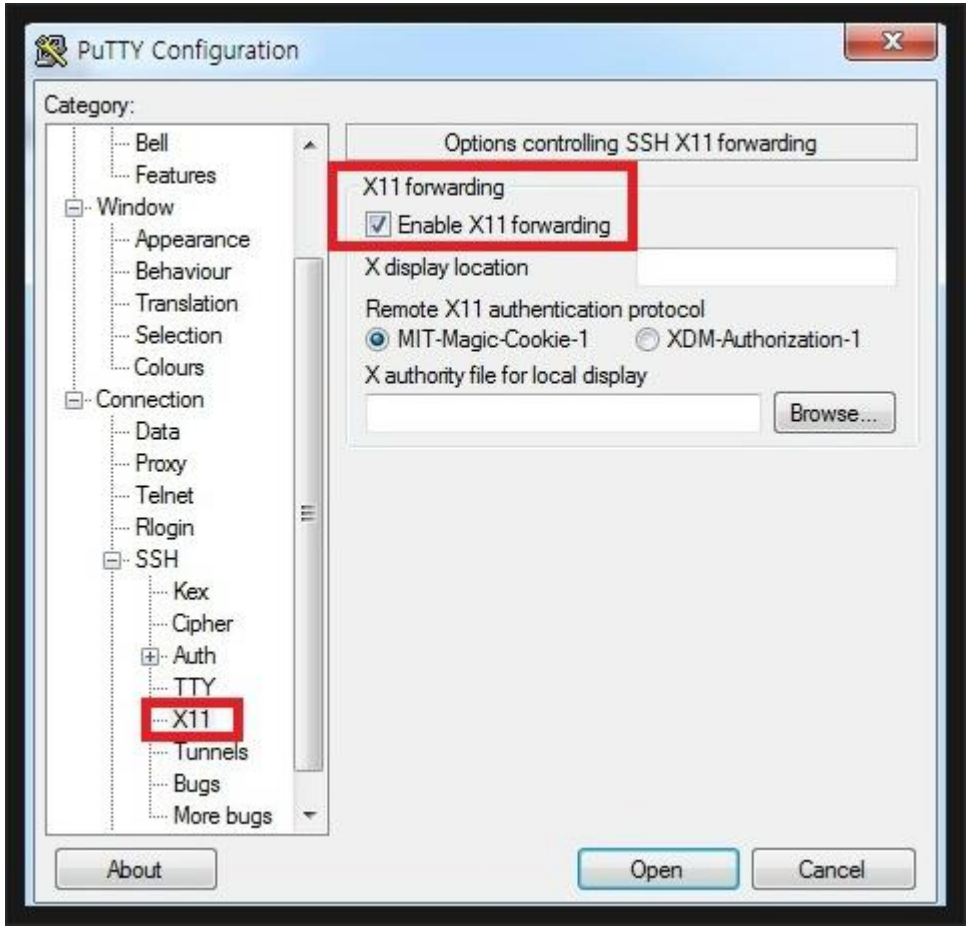
- Xming
  - https://sourceforge.net/projects/xming/



  - Install & Launch Xming Default Setting

X Window Setting @ PuTTY

- Configuration → Connection → SSH → X11
  - **Enable X11 forwarding** checkbox



Copy GLX Header & Skeleton files

Copy XWindow.h file to your project folder

Order1: cp /home/share/XWindow.h ./

Copy XWindow.cpp file to your project folder

Order2: cp /home/share/XWindow.cpp ./

```
#include XWindow.h

int main()
{
    Code……..


}
```

Create an Empty Window @ GPU Server

```
#include "XWindow.h"

Display                 *dpy;
Window                  root, win;
GLint                   att[] = { GLX_RGBA, GLX_DEPTH_SIZE, 24, GLX_DOUBLEBUFFER, None };
XVisualInfo             *vi;
Colormap                cmap;
XSetWindowAttributes    swa;
GLXContext              glc;
XWindowAttributes       gwa;
XEvent                  xev;

int main(int argc, char *argv[]) {
```

Settings environment

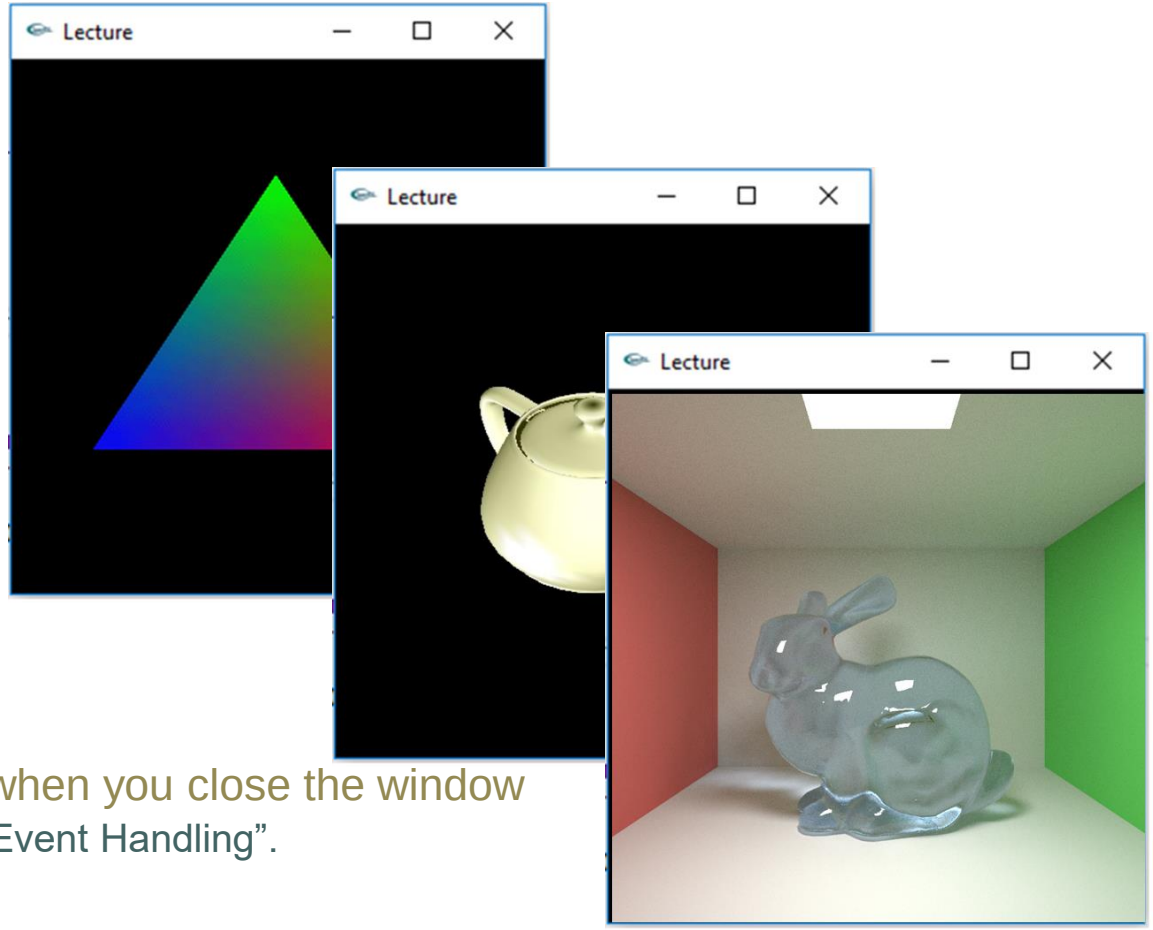
Create Windows

```
dpy = XOpenDisplay(NULL);
if(dpy == NULL) {
                        printf("\n\tcannot connect to X server\n\n");
                        exit(0);
}
root = DefaultRootWindow(dpy);

vi = glXChooseVisual(dpy, 0, att);
if(vi == NULL) {
                        printf("\n\tno appropriate visual found\n\n");
                        exit(0);
}
else {
                        printf("\n\tvisual %p selected\n", (void *)vi->visualid); /* %p creates hexadecimal output like in glxinfo */
}
cmap = XCreateColormap(dpy, root, vi->visual, AllocNone);
swa.colormap = cmap;
win = XCreateWindow(dpy, root, 0, 0, 600, 600, 0, vi->depth, InputOutput, vi->visual, CWColormap | CWEventMask, &swa);
XMapWindow(dpy, win);
XStoreName(dpy, win, "Lecture");
glc = glXCreateContext(dpy, vi, NULL, GL_TRUE);
glXMakeCurrent(dpy, win, glc);
glEnable(GL_DEPTH_TEST);
```

Drawing Something

- **GLX provides no main loop function**
  - Let us simply use an infinite loop for now

```
void display() {
        glClear(GL_COLOR_BUFFER_BIT
         | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        //Draw Call Here
        glXSwapBuffers(dpy, win);
}

int main(){
        while(1)
        {
            display();
        }
}
```



  - Caution
    - There is error call text when you close the window
      - We will say this at “Event Handling”.

Assignment #1

Goal of Assignment

1. Programming `vectorAdd()` on the GPU
   - Skeleton code is given
     - Including Programming Flow Guide
   - Must run `vectorAdd()` function on GPU
2. Draw Solid Teapot in an OpenGL Window
   - Skeleton code is given
     - Including OpenGL initialization routines
   - Call “Drawing **Solid Teapot**” function
   - Change the title of window to your student ID and name
     - Ex) 2015000000 박지혁

※ **Submission Due Date : 09/23**

Compile and Run on GPU Server

Both assignments must run on GPU Server

1. Connect to GPU server

2. Transfer your code to GPU server or write code on GPU Server

3. Compile and run your assignment

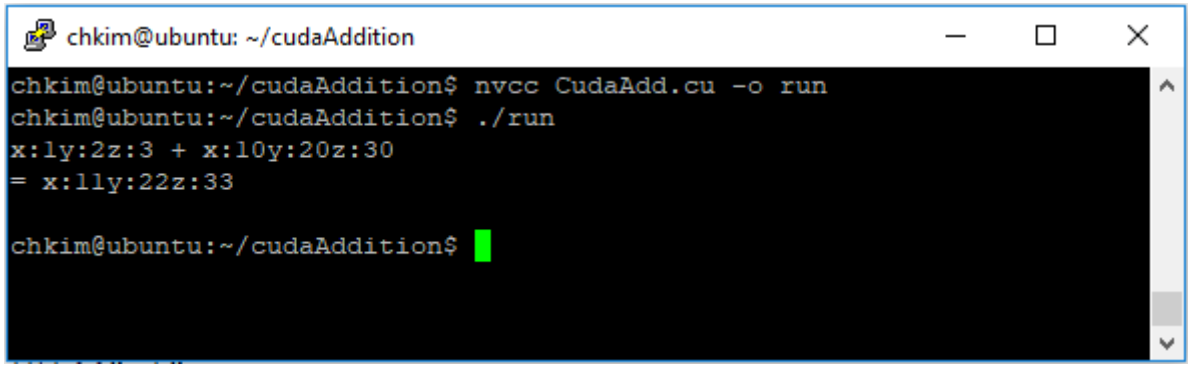
4. Submit your source code and running image

CUDA Assignment

- **Programming `vectorAdd()` on GPU**

$$\vec{A} + \vec{B} = \vec{C}$$

  - `vectorAdd()` function must run on the GPU
  - Compile and print result on GPU Server



Write a Device Kernel Function

- **Add two 3D vectors**
  - Use 3D vector structure.

```
struct vec3
{
    float x,y,z;

};
```

- **Function must run on the GPU**

CUDA Assignment

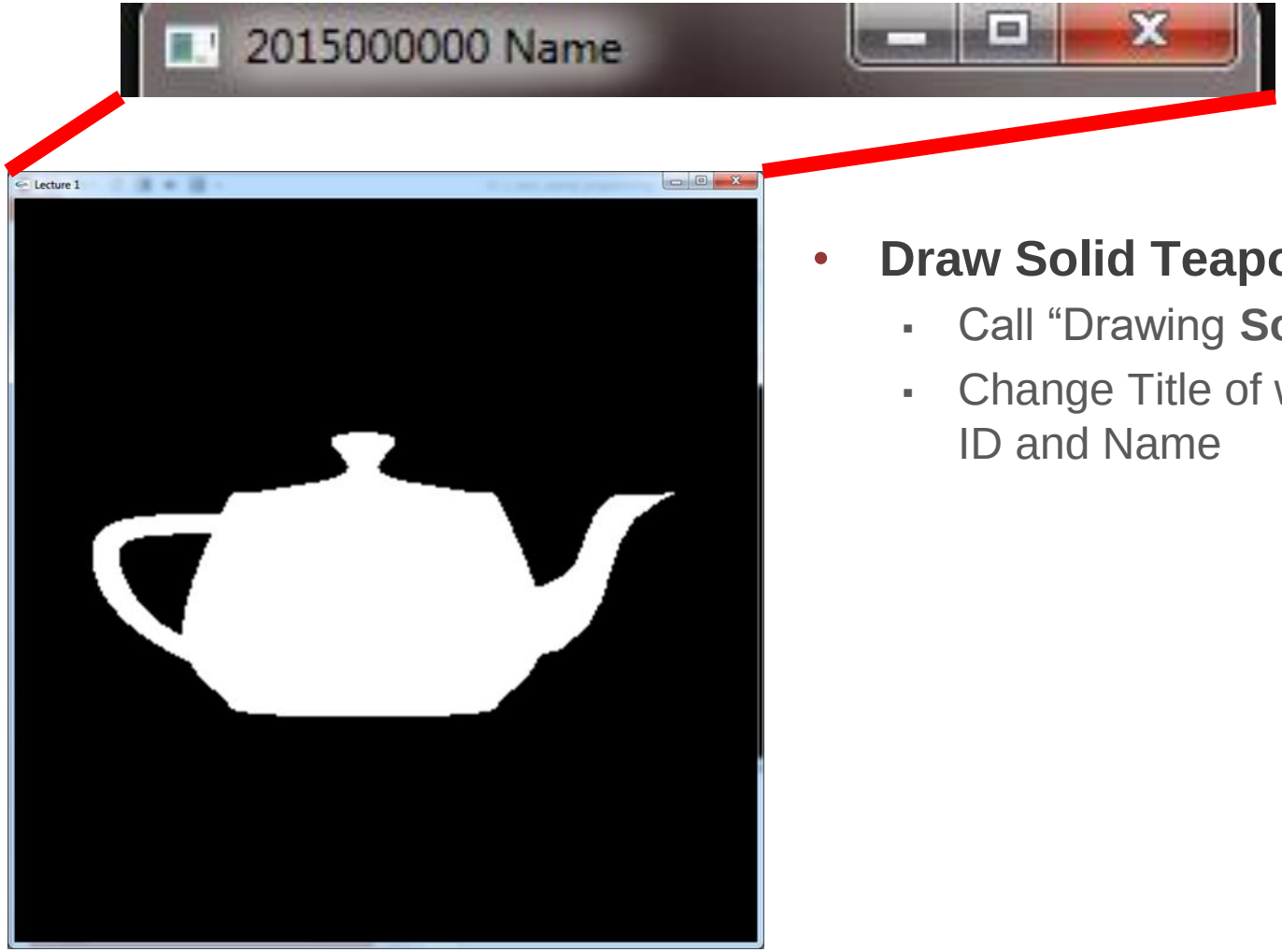
Skeleton Code (1)

```
#include "cuda_runtime.h"
#include <stdio.h>
#include <stdlib.h>


__global__ void vectorAdd(/*in&out arguments*/)
{
	int tid = threadIdx.x;
	/*		1-1. write vector addition function		*/
}


int main( void )
{
	/*		2-1. Check whether a proper device is mounted		*/

	/*		2-2. Declare Host and Device pointer variables		*/

	/*		2-3. Allocate Host memory		*/

	/*		2-4. Allocate Device memory		*/
```

Skeleton Code (2)

```
    /*    2-5. Check that memory is allocated well on Device                */

    /*    2-6. Setup Input values to host array                             */

    /*    2-7. Copy memory for Input array from Host to Device              */

    /*              2-8. Call Kernel Function with <<<1, 1>>>               */
    vectorAdd<<<1,1>>>(/*in&out arguments*/);

    /*    2-9. Copy memory for Result from Device to Host                   */

    /*    2-10. Print Results                                               */

    /*    2-11. Release Host and Device memory                              */

    return 0;

}
```

OpenGL Assignment



- **Draw Solid Teapot on the X Window**
  - Call “Drawing **Solid Teapot**” Function
  - Change Title of window to your Student ID and Name

OpenGL Assignment

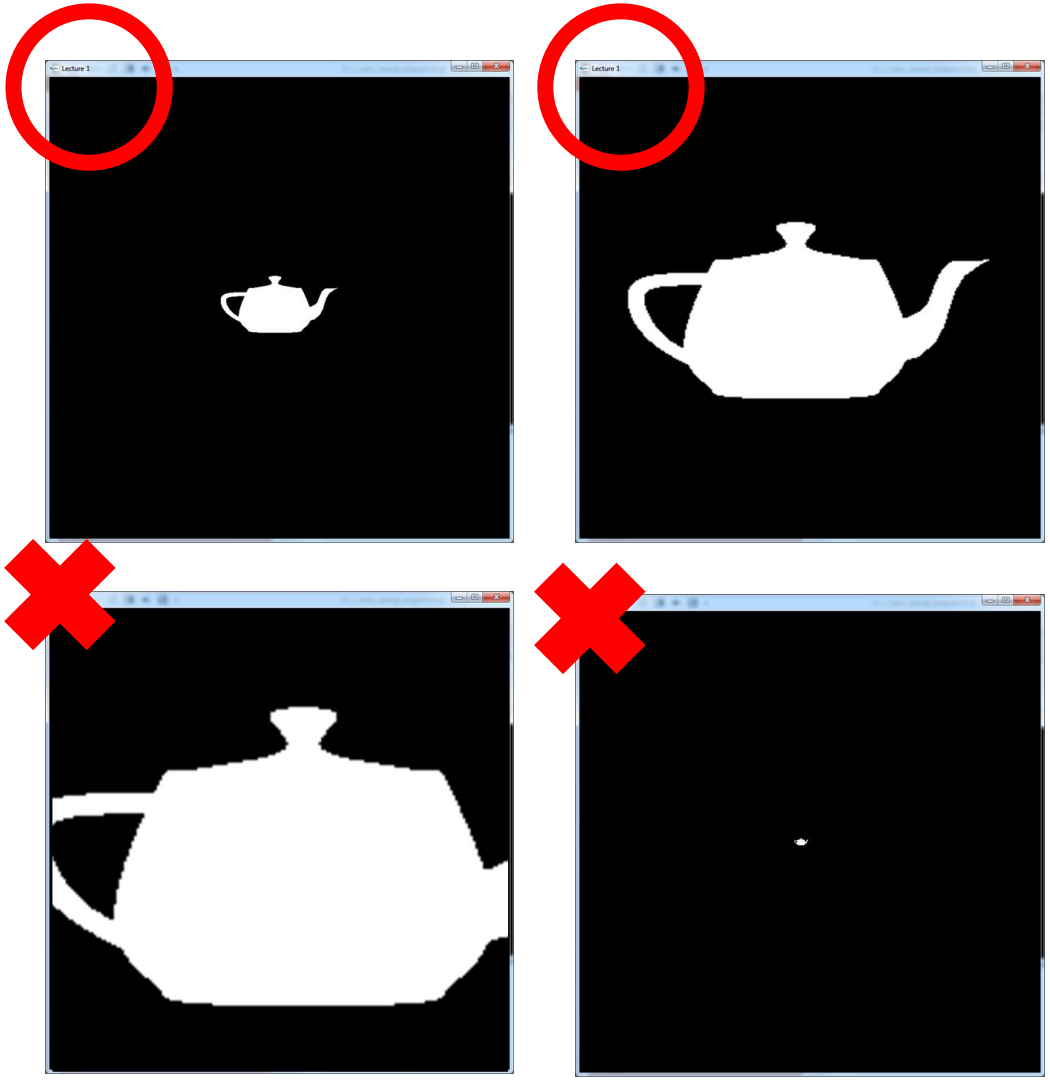
Call “Drawing Solid Teapot” Function

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
        //Draw Call Here
    glXSwapBuffers(dpy, win);
}

int main(int argc, char *argv[])
{
        /*Create Window*/

        while(1)
        {
            display();
        }
}
```

- **Write draw function**
  - You can draw solid teapot with two(**Draw, Initialize)** function.
    - Hint: **GLUT**

Caution

- Whole part of Teapot must be on the Window
- Teapot must be **visible** size



Submit the Assignment

- **Submit the zip file @ Blackboard**
  - File name must be “Assignment1_StudentID_Name.zip”
    - Ex. Assignment1_2015000000_박지혁.zip
  - Zip file must include two folder
    - CUDA
    - OpenGL
  - Each assignment folder must include
    - Src file
    - Result running Image file