Getting Started (1)

GPU Programming

2022학년도
2학기

ⓒ nvidia

OpenGL

OpenGL이란?

- OpenGL은 대표적인 그래픽스 API (application programming interface) 중 하나로 여겨짐
  - Khronos Group이라는 비영리 컨소시엄에서 관리
  - 보통 API는 명세 (specification) 또는 구현 (implementation)을 모두 지칭
  - OpenGL은 명세만 제공 – 각 함수의 출력 및 기능만 정의

- OpenGL의 구현은 GPU vendor가 맡음
  - GPU에 따라 OpenGL 지원 버전 및 구현 방법도 달라짐.
  - 드라이버 업데이트를 통해, bug fix 및 성능 개선, 추가 기능 구현 가능
  - OpenGL을 S/W로 구현한 라이브러리도 존재 Home — The Mesa 3D Graphics Library

- 본 과목에서 사용하는 OpenGL 버전은 3.3임
  - OpenGL 3.3 (Core Profile) - March 11, 2010 (khronos.org)

- OpenGL 공식 사이트: OpenGL - The Industry Standard for High Performance Graphics

OpenGL의 역사 및 버전

- OpenGL
  - 1.0 (1992)~1.5 (2003) – fixed function
  - 2.0 (2004)~2.1 (2006) – early programmable (vertex & fragment shader)
  - 3.0 (2008)~4.6 (2017) – modern programmable (geometry shader, tessellation 등 지원 추가)

- OpenGL ES
  - 모바일/임베디드 시스템을 위해 OpenGL에서 일부 기능을 추가 및 제거한 버전
  - 1.0 (2003)~1.1(2004) – fixed function (OpenGL 1.3~1.5에 대응)
  - 2.0 (2007) – early programmable (OpenGL 2.0~3.0 대응)
  - 3.0 (2012)~3.2(2015) – modern programmable (OpenGL 4.3에 대응)

Core-profile vs Immediate mode

- Immediate mode
  - 예전 fixed function pipeline 시절 주로 쓰던 방법
  - 코딩이 쉽지만 효율이 낮은 구식(deprecated) 방법
  - OpenGL Core-profile 사용시,
    또는 OpenGL ES 2.0 이상 사용시 지원되지 않음

- Core-profile
  - 유연하고 효율적이며,
    OpenGL이 실제로 어떻게 작동하는지 파악하기 쉬움
  - 코딩은 훨씬 더 복잡해짐

What does "immediate mode" mean in OpenGL? - Stack Overflow

```
glBegin(GL_TRIANGLES);
    glColor3f(1.0f, 0.0f, 0.0f);   glVertex2f(0.0f,   1.0f);
    glColor3f(0.0f, 1.0f, 0.0f);   glVertex2f(0.87f,  -0.5f);
    glColor3f(0.0f, 0.0f, 1.0f);   glVertex2f(-0.87f, -0.5f);
glEnd();
```

```
float verts = {...};
float colors = {...};
static_assert(sizeof(verts) == sizeof(colors), "");

// not really needed for this example, but mandatory in core profile after GL 3.2
GLuint vao;
glGenVertexArrays(1, &vao);
glBindVertexArray(vao);

GLuint buf[2];
glGenBuffers(2, buf);

// assuming a layout(location = 0) for position and
// layout(location = 1) for color in the vertex shader

// vertex positions
glBindBuffer(GL_ARRAY_BUFFER, buf[0]);
glBufferData(GL_ARRAY_BUFFER, sizeof(verts), verts, GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);

// copy/paste for color... same code as above. A real, non-trivial program would
// normally use a single buffer for both -- usually with stride (5th param) to
// glVertexAttribPointer -- that presumes interleaving the verts and colors arrays.
// It's somewhat uglier but has better cache performance (ugly does however not
// matter for a real program, since data is loaded from a modelling-tool generated
// binary file anyway).
glBindBuffer(GL_ARRAY_BUFFER, buf[1]);
glBufferData(GL_ARRAY_BUFFER, sizeof(colors), colors, GL_STATIC_DRAW);
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, 0);

glDrawArrays(GL_TRIANGLES, 0, 3);
```

Extensions

- 기존 OpenGL 스펙에 없는 새로운 기술이 등장하면, 일부 플랫폼 또는 GPU에서 이를 확장 지원하기 위해 extension을 사용

- 특정 GPU 또는 플랫폼에서만 동작하므로, 개발자는 이를 적절히 처리할 수 있도록 코딩해야 함

```
if(GL_ARB_extension_name) {
    // Do cool new and modern stuff supported by hardware
}
else {
    // Extension not supported: do it the old way
}
```

Extension Prefix	Extension Vendor
ARB	OpenGL® ARB approved extensions.
NV	NVIDIA Corporation.
NVX	NVIDIA Corporation. Experimental extension.[1]
ATI	ATI Technologies, Inc.
3DLABS	3DLABS, Inc.
SUN	Sun Microsystems, Inc.
SGI	Silicon Graphics, Inc.
SGIX	Silicon Graphics, Inc. Experimental extension.[1]
SGIS	Silicon Graphics, Inc. Experimental extension.[1]
INTEL	Intel Corporation.
3DFX	3dfx Interactive.
IBM	International Business Machines Corporation, or simply IBM.
MESA	The Mesa 3D Graphics Library.
GREMEDY	Graphic Remedy, Ltd.
OML	Khronos Group, Inc. API: OpenML®
OES	Khronos Group, Inc. API: OpenGL® ES
PGI	Portland Group Inc.
I3D	Intense3D, now 3DLABS Inc.
INGR	Intergraph Corporation.
MTX	Matrox Electronic Systems Ltd.
Extension Prefix	**Extension Platform**
WGL_EXT, WGL_ARB, WGL_ATI, WGL_NV	Microsoft Corporation. WGL = **W**indows Open**GL**
GLX_EXT, GLX_ARB, GLX_ATI, GLX_NV	X-Window-based platforms.
AGL	Apple Inc. AGL = **A**pple Open**GL**

OpenGL Extension - OpenGL Wiki (khronos.org)

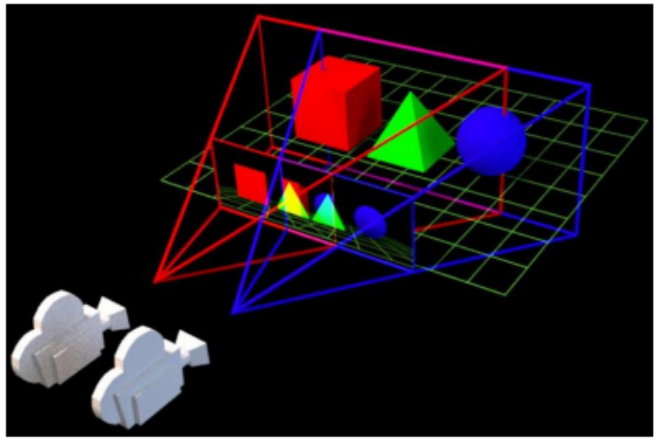
Extensions

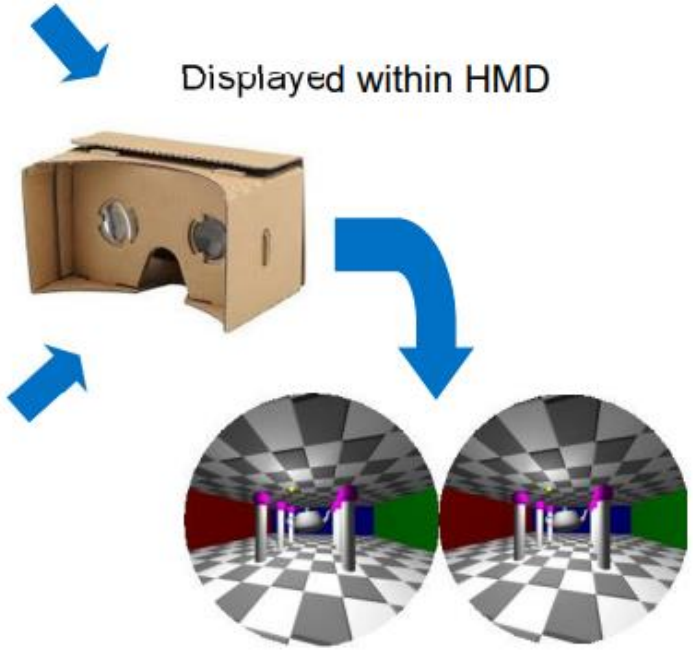
OpenGL Extensions Used in LMS VR Pipeline

Pascal's NV_stereo_view_rendering Extension

- Allows vertex shader to output two clip-space positions
  - (x_1,y,z,w) and (x_2,y,z,w)
  - Results in TWO primitives one for left eye & one for right eye
- New GLSL built-ins
  - gl_SecondaryPositionNV
    - Like gl_Position but for "second eye's view"
  - gl_SecondaryViewportMaskNV[]
    - Like gl_ViewportMaskNV[] but for "second eye's view"
- Also can steer primitives to different texture array slices
  - layout(secondary_view_offset = 1) int gl_Layer;

SINGLE PASS STEREO



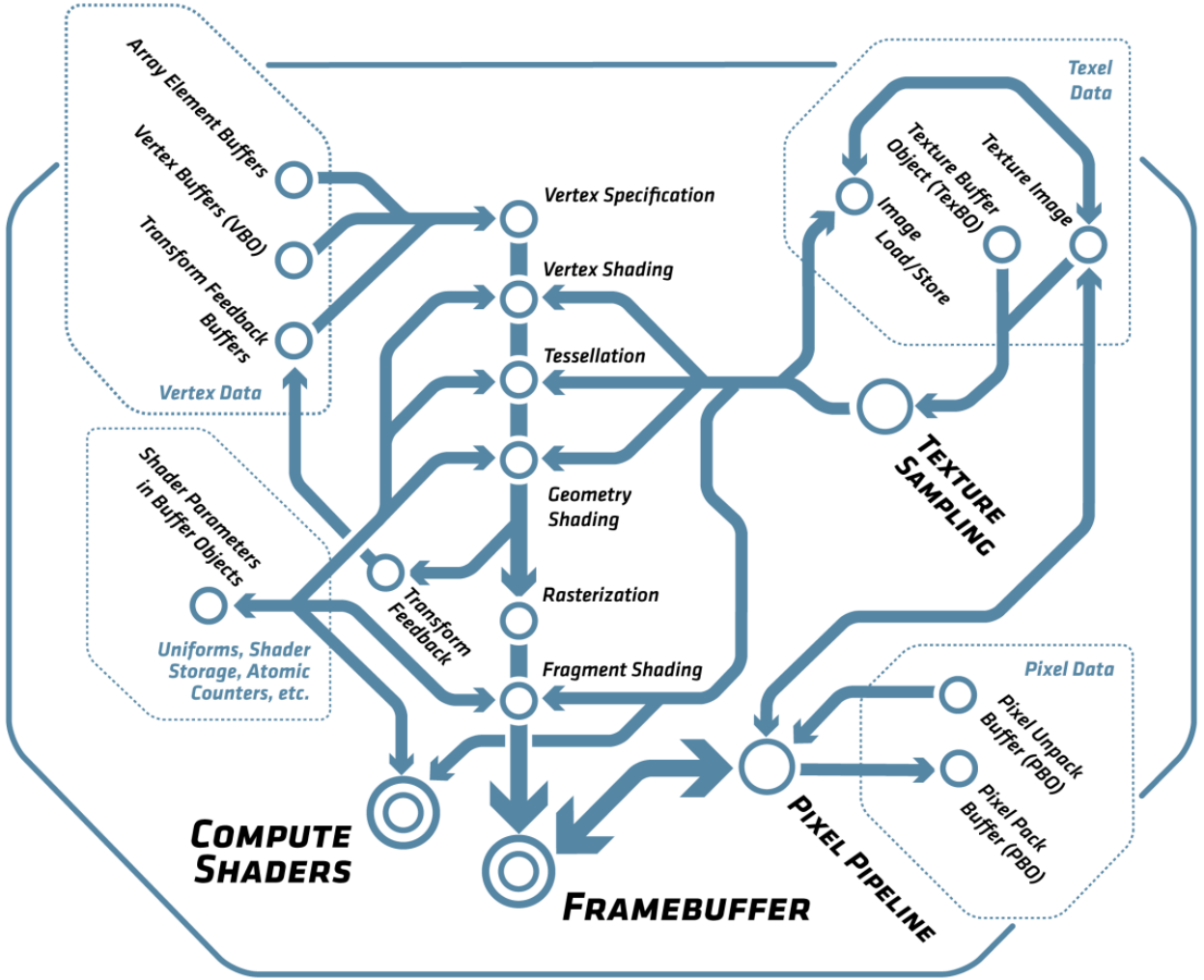


NVIDIA OpenGL in 2016 (gputechconf.com)

State Machine

- OpenGL은 상태기계 (state machine)임
  - OpenGL이 현재 동작하는 상태를 정의하는 변수들의 집합
  - 이 상태를 OpenGL context라고 지칭

- State-changing functions & state-using functions
  - OpenGL에는 옵션 설정이나 버퍼 조작을 통해 이 상태를 바꾸는 함수들과,
  - 이와 같이 바뀐 context를 통해 렌더링을 수행하는 함수들로 구성



OpenGL 4.6 (Core Profile) - May 5, 2022 (khronos.org)

Objects

- 기본적으로 OpenGL 라이브러리는 C로 작성
  - 단, 많은 파생(JAVA 등)도 허용
  - C 언어의 언어 구조가 다른 고급언어와 일치하기 않으므로, OpenGL은 몇가지 추상화를 염두에 두고 개발

- OpenGL의 객체
  - OpenGL 상태의 일부(subset)를 표현하는 옵션들의 모음
  - 하나의 프로그램에서 여러 개의 객체를 사용 가능
  - OpenGL 상태를 사용하는 작업을 시작할 때마다 객체를 컨텍스트에 바인딩(binding)

Objects

- OpenGL 객체 스타일의 코딩 예시 (실제 OpenGL 함수는 아님)

```
struct object_name {
    float  option1;
    int    option2;
    char[] name;
};
```

```
// The State of OpenGL
struct OpenGL_Context {
    ...
    object_name* object_Window_Target;
    ...
};
```

```
// create object
unsigned int objectId = 0;
glGenObject(1, &objectId);
// bind/assign object to context
glBindObject(GL_WINDOW_TARGET, objectId);
// set options of object currently bound to GL_WINDOW_TARGET
glSetObjectOption(GL_WINDOW_TARGET, GL_OPTION_WINDOW_WIDTH, 800);
glSetObjectOption(GL_WINDOW_TARGET, GL_OPTION_WINDOW_HEIGHT, 600);
// set context target back to default
glBindObject(GL_WINDOW_TARGET, 0);
```

ⓒ nvidia

Creating a Window

준비 사항

- 윈도우일 경우
  - Visual Studio Community 설치 (C++를 사용한 데스크톱 개발 체크표시)
  - CMake 설치

- 리눅스일 경우
  - apt-get install g++ cmake git
  - 이후 https://github.com/JoeyDeVries/LearnOpenGL의 Linux building 참조

- Mac일 경우
  - Xcode에서 Modern OpenGL 사용하기 (GLFW, GLEW) : 네이버 블로그 (naver.com)

GLFW

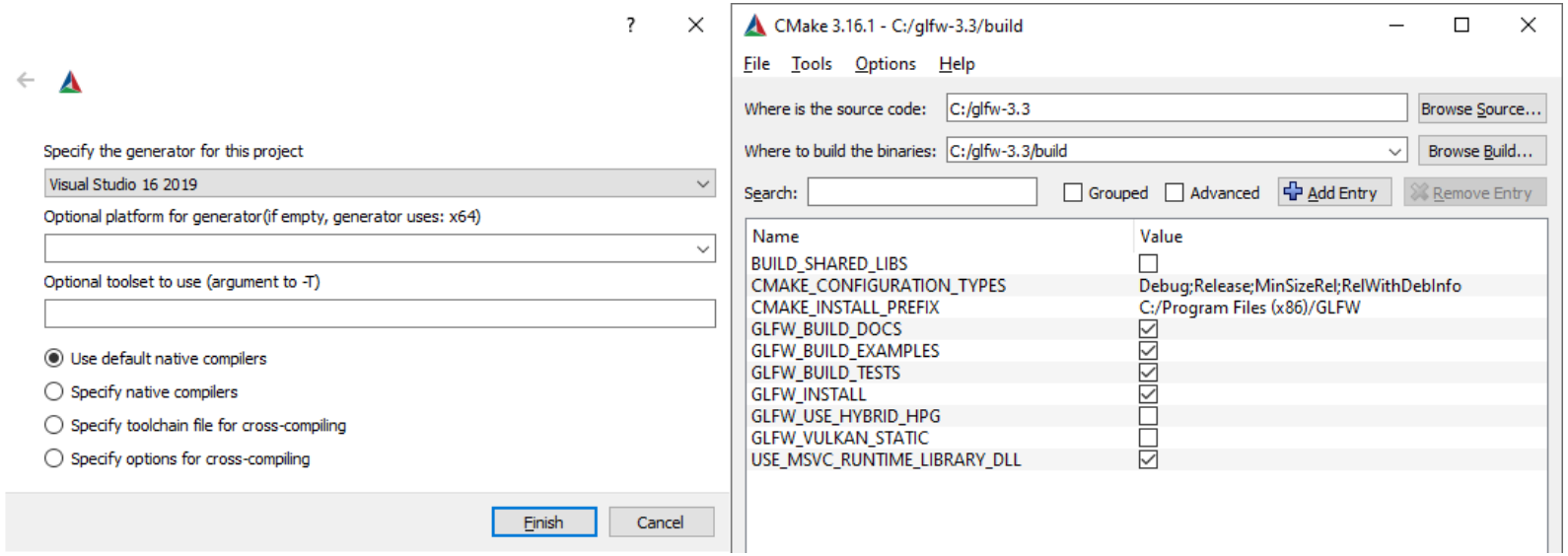
- OpenGL 사용을 쉽게 해 주는 라이브러리 중 하나
  - 화면에 무언가를 그리는 데 필요한 것들, OpenGL context 생성, Window 파라미터 정의, 사용자 입력 처리 등 다양한 기능 제공
  - 다른 라이브러리로는 GLUT, SDL, SFML 등이 있음

- Visual Studio (VS) & 윈도우 기준으로 빌드 방법 설명

- GLFW 다운로드
  - 다운로드 사이트: https://www.glfw.org/download.html
  - 별도 빌드 대신 미리 컴파일된 라이브러리도 사용 가능 (단, 호환은 보장 못함)

GLFW

- CMake를 사용하여 VS 프로젝트 생성 (Configure → Generate)



- VS 프로젝트 생성 후, VS에서 GLFW.sln을 열어 build
  - src/Release에 glfw3.lib를 생성한 후, OpenGL 관련 프로젝트 (또는 별도 OpenGL 폴더)의 lib 폴더에 복사
  - GLFW의 include 폴더를 통째로 OpenGL 관련 프로젝트 (또는 별도 OpenGL 폴더)의 include 폴더에 복사

GLAD

- 실제 OpenGL 함수들의 위치
  - 이는 드라이버마다 다르므로 컴파일시에는 알 수 없고, 런타임시에 질의해야 함
  - 이 위치는 함수 포인터에 저장하게 되고, 위치 검색 방법은 운영체제마다 다름

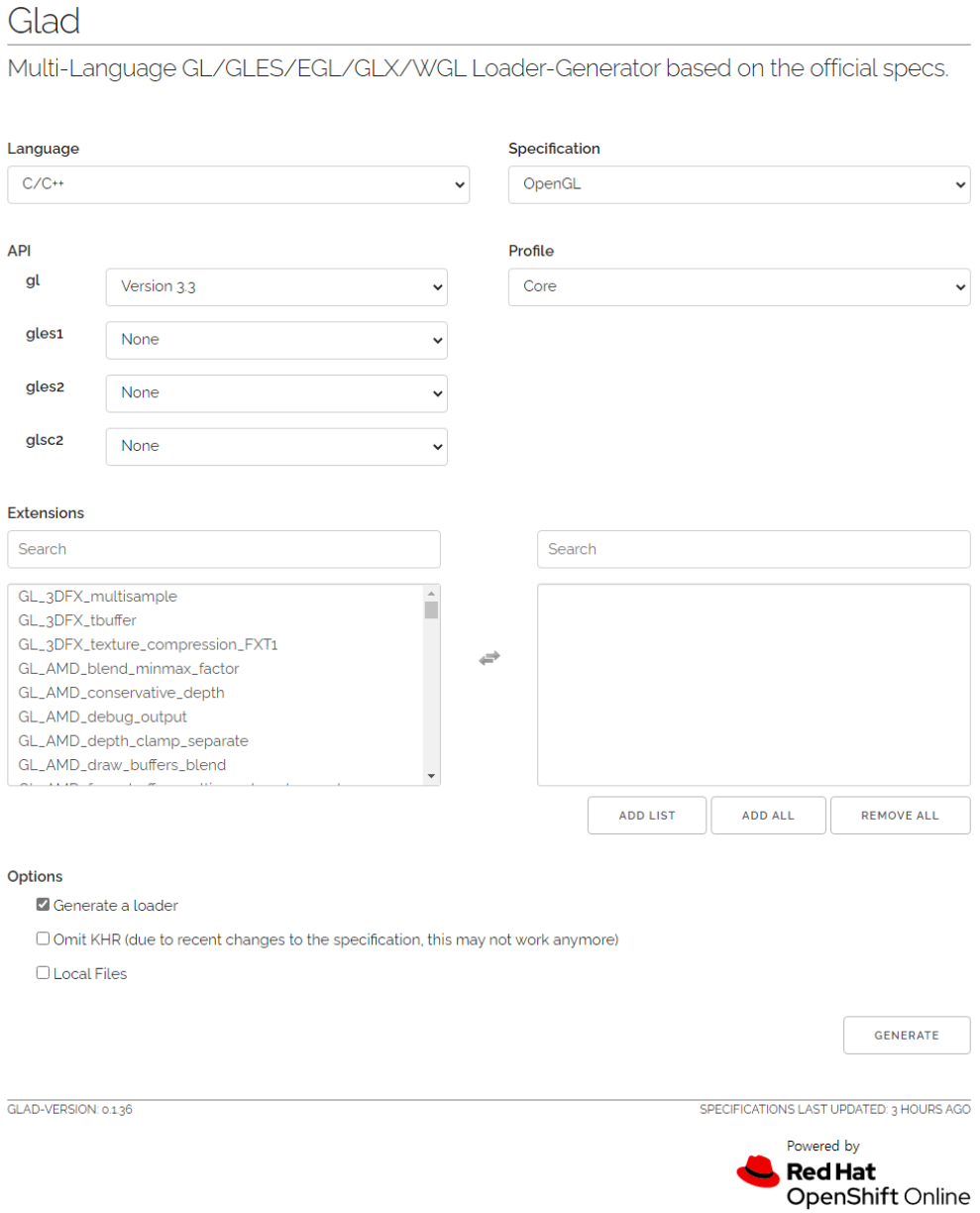
- 윈도우에서의 함수 포인터 사용 예

```
// define the function's prototype
typedef void (*GL_GENBUFFERS) (GLsizei, GLuint*);
// find the function and assign it to a function pointer
GL_GENBUFFERS glGenBuffers  = (GL_GENBUFFERS)wglGetProcAddress("glGenBuffers");
// function can now be called as normal
unsigned int buffer;
glGenBuffers(1, &buffer);
```

- GLAD를 사용하면 이러한 번거로움을 없앨 수 있음

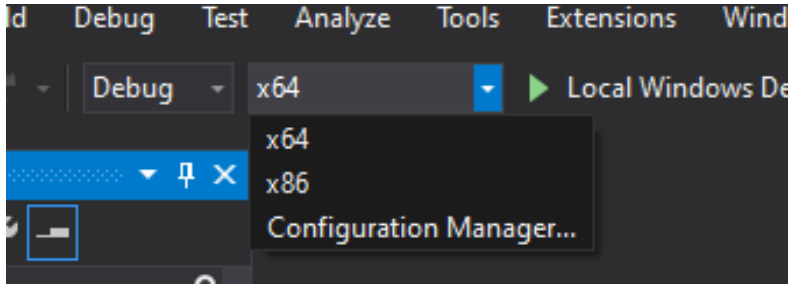
GLAD

- https://glad.dav1d.de 로 이동
  - 3.3 & Core 선택 후 Generate
  - 생성된 zip 파일 다운로드
  - Zip 파일 내의 src, include 폴더를 OpenGL 프로젝트에 복사 후 등록

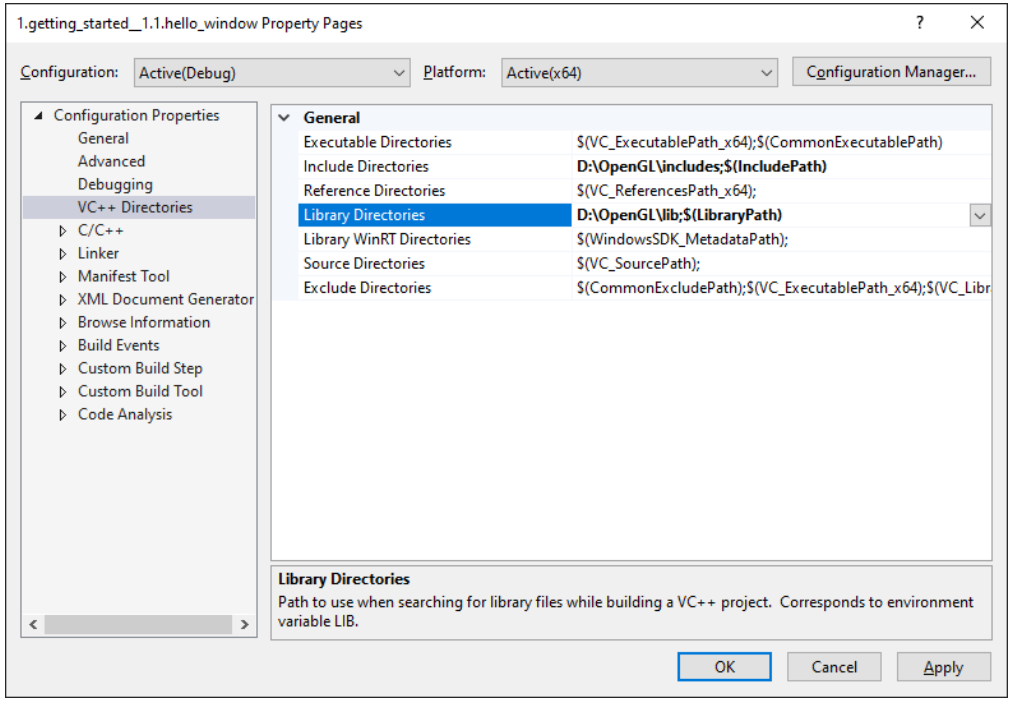


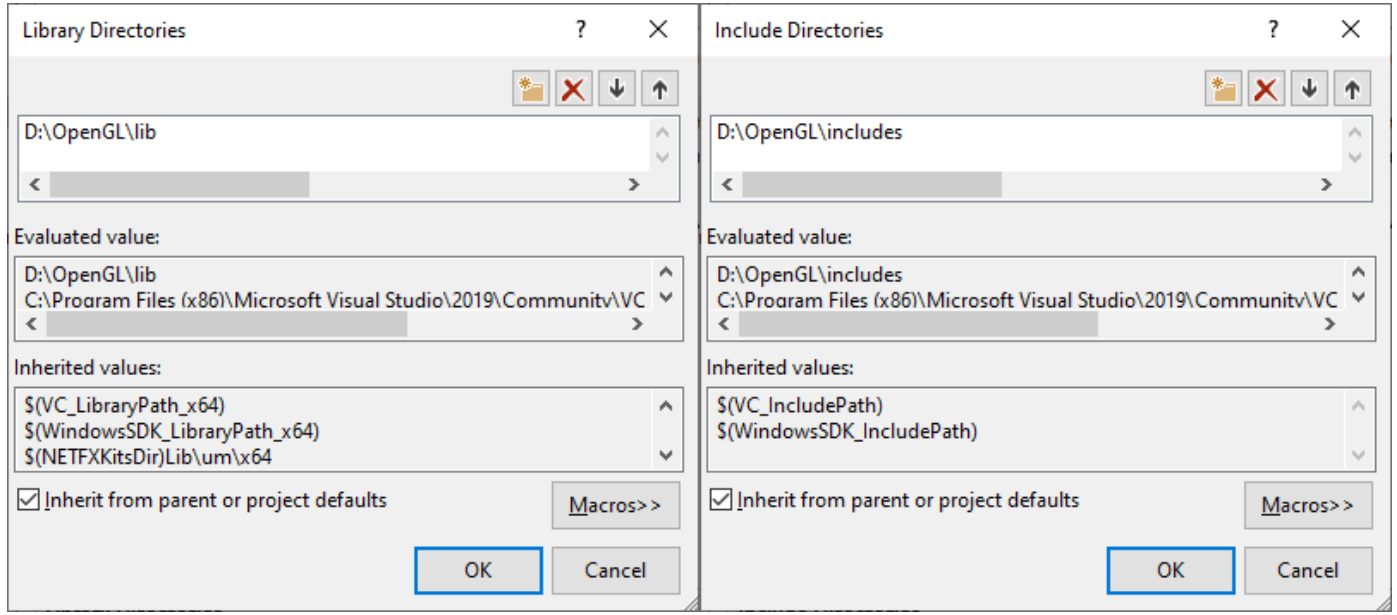
새 VS 프로젝트 생성

- 새 빈 프로젝트 (C++) 생성
  - 빌드 플랫폼은 x64 선택



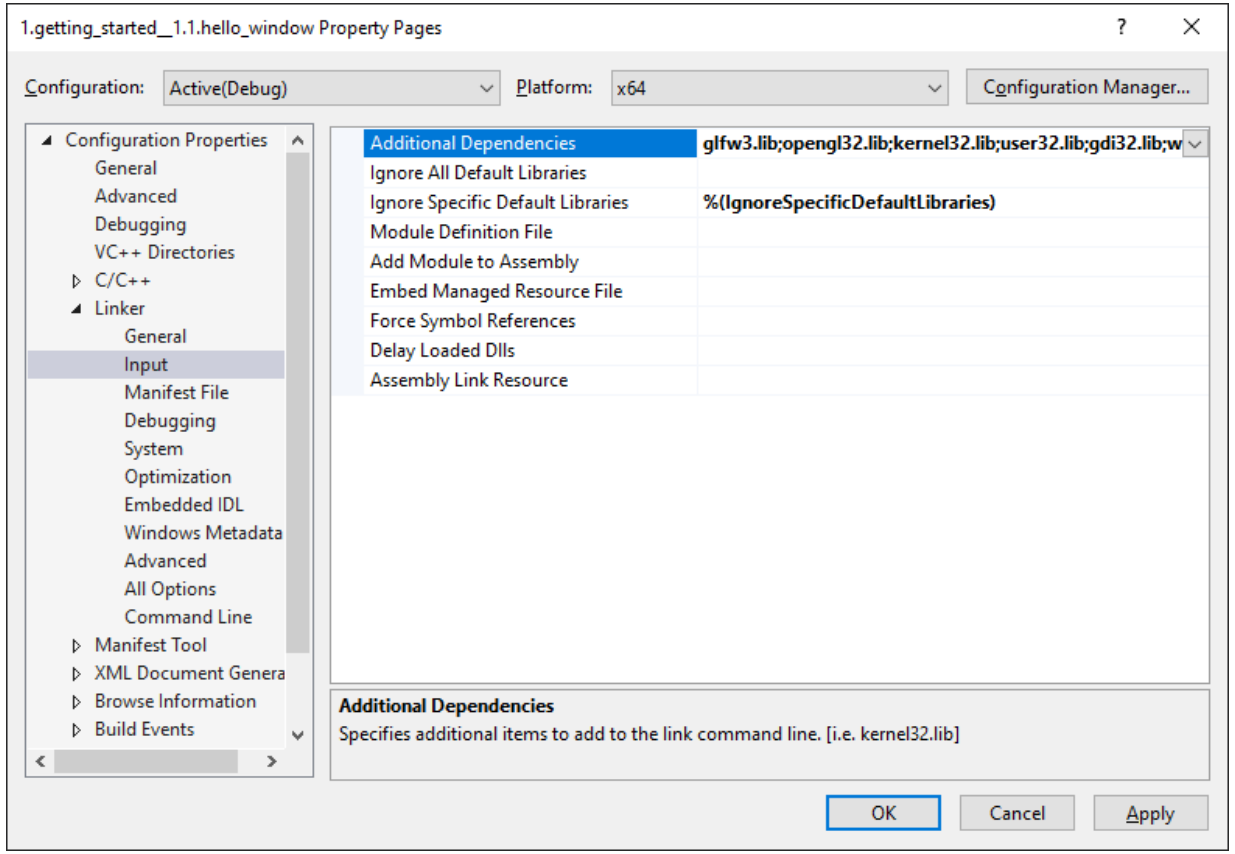
- Linking 설정
  - 상대 또는 절대 경로로 includes 및 lib 폴더 설정





새 VS 프로젝트 생성

- Linking 설정 (cont.)
  - 추가 종속성에 glfw3.lib; opengl32.lib; 추가
  - openGL32.lib는 MS SDK에 이미 포함되어 있음



- 이제 새 cpp 파일을 만들고 아래 코드 입력 가능

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>
```

ⓒ nvidia

Hello Window

Hello Window

- 전체 코드는 아래 페이지에서 다운 가능
  - Code Viewer. Source code: src/1.getting_started/1.2.hello_window_clear/hello_window_clear.cpp (learnopengl.com)

- 메인 함수
  - GLFW 초기화 및 OpenGL 버전 선택
  - 마지막 주석처리 된 행은 Mac에서만 주석 해제

```
int main()
{
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    //glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);

    return 0;
}
```

Hello Window

- 윈도우 객체 생성 부분 추가

```
GLFWwindow* window = glfwCreateWindow(800, 600, "LearnOpenGL", NULL, NULL);
if (window == NULL)
{
    std::cout << "Failed to create GLFW window" << std::endl;
    glfwTerminate();
    return -1;
}
glfwMakeContextCurrent(window);
```

- GLAD 초기화

```
if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
{
    std::cout << "Failed to initialize GLAD" << std::endl;
    return -1;
}
```

Hello Window

- Viewport 설정
  - Viewport는 OpenGL에서 렌더링할 윈도우를 의미
  - 사용자가 윈도우 크기를 바꿀 경우를 대비해,
    callback 함수를 만들고 윈도우 크기가 바뀔 때마다 이를 호출되도록 함

```
void framebuffer_size_callback(GLFWwindow* window, int width, int height)
{
    glViewport(0, 0, width, height);
}
```

```
glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
```

  - 이 callback함수에서는 glViewport() 함수를 호출하며,
    이 함수는 [-1, 1]로 정의되어 있는 NDC (normalized device coordinates)를 윈도우 좌표계로 변환

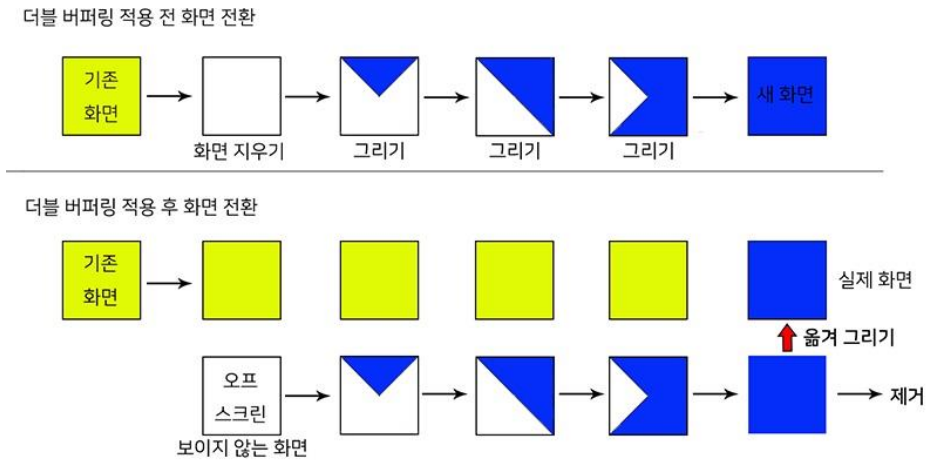
Callback 함수란?
- 다른 함수의 인자로 사용되는 함수
- 이벤트에 의해 호출 가능

Ready Your Engines

- Render loop 설정
  - 프로그램 종료 전까지 계속 이미지를 그림(swap buffer)
  - 사용자 입력 처리 (poll events)
  - 랜더 루프가 한 번 반복되면 이를 frame이라고 지칭

```
while(!glfwWindowShouldClose(window))
{
    glfwSwapBuffers(window);
    glfwPollEvents();
}
```

- Double buffering



Double Buffering - 더블 버퍼링 기법 (tistory.com)

single buffering　　double buffering

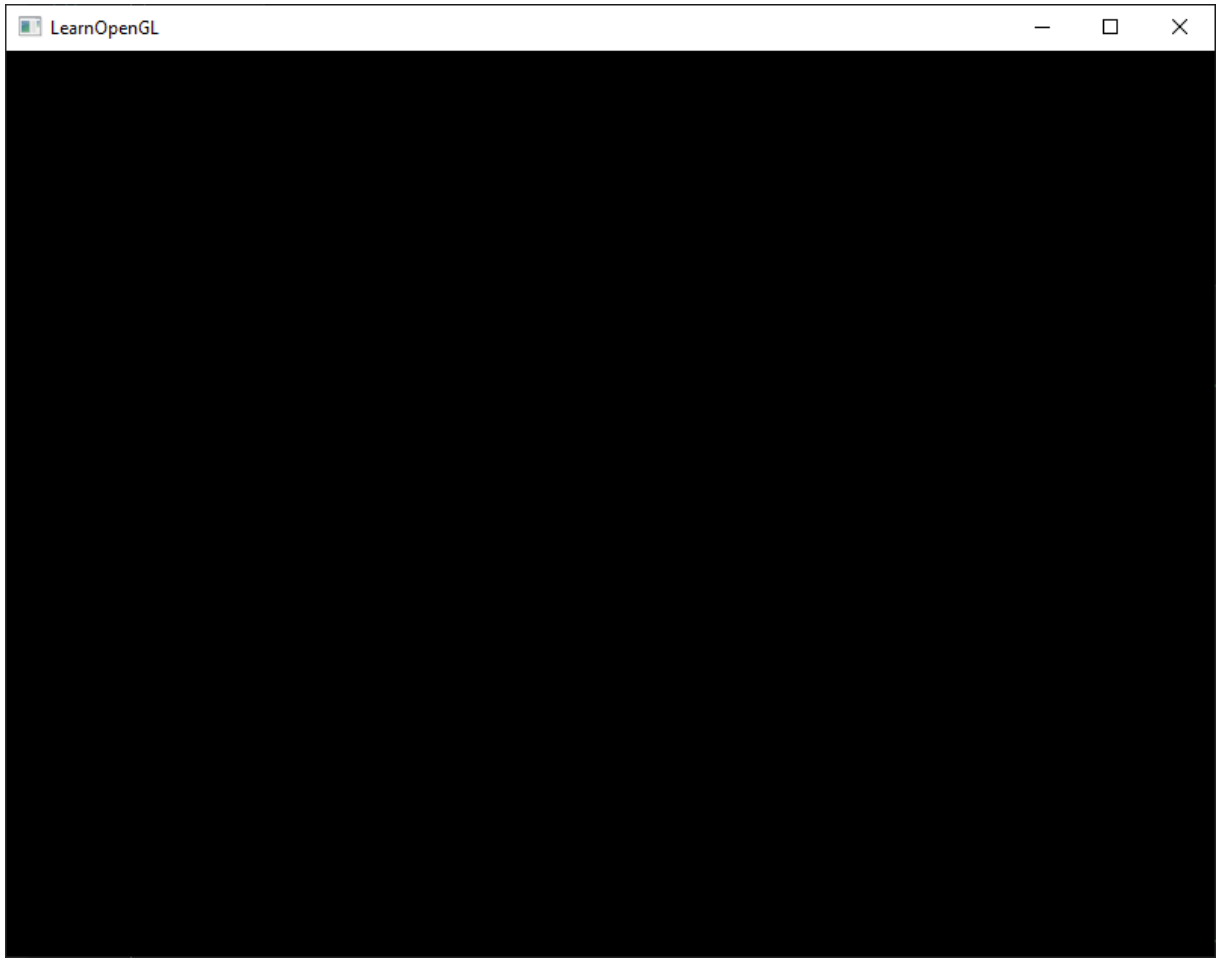
Yoctopuce displays and double buffering

One Last Thing

- 렌더 루프 종료시 GLFW 리소스 삭제

```
glfwTerminate();
return 0;
```

- 앞서 소개한 코드를 모두 타이핑했으면 오른쪽과 같은 빈 윈도우를 볼 수 있음



Input

- 키 입력을 처리하는 함수 추가
  - 아래 예시는 Esc를 누르면 프로그램을 종료하도록 함

```
void processInput(GLFWwindow *window)
{
    if(glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
}
```

- 렌더 루프에 위 함수 호출을 추가

```
while (!glfwWindowShouldClose(window))
{
    processInput(window);

    glfwSwapBuffers(window);
    glfwPollEvents();
}
```

Rendering

- 렌더 루프 안에 화면 색깔 바꾸는 코드를 추가

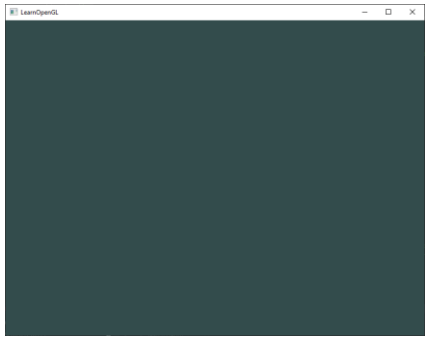
```
// render loop
while(!glfwWindowShouldClose(window))
{
    // input
    processInput(window);

    // rendering commands here
    ...

    // check and call events and swap the buffers
    glfwPollEvents();
    glfwSwapBuffers(window);
}
```

state-setting function

```
glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
```

state-using function

- 실행 결과



ⓒ nvidia

마무리

마무리

- 이번 시간에는 아래와 같은 내용을 살펴보았습니다.
  - OpenGL 개요
  - GLFW, GLAD 설정 방법
  - OpenGL을 이용하여 윈도우를 만들고 렌더 루프를 설정하는 방법
  - 콜백함수 사용법, 더블 버퍼링의 원리

- 실습 문제
  - F1~F6키를 눌러서, 윈도우에 칠해진 RGB값을 증가 또는 감소시키도록 해 보세요.

- 다음 주차에는 아래 내용을 다룹니다.
  - Hello Triangle, Shaders & Textures
  - 목요일에 코드 일부 및 관련 이론 내용을 설명하고, 화요일에 전체 코드를 직접 실습해보는 방향으로 진행