openGL 기초

2024-2 컴퓨터 그래픽스

OpenGL 소개

- OpenGL의 역사
  - 1982년 Silicon Graphics(SGI)사의 워크스테이션용 그래픽스 라이브러리 아이리스 지엘 (Iris GL) 로 시작하여 1992년 OpenGL 1.0으로 출시된 2D 및 3D 그래픽스 프로그램 개발 업계 표준 API
  - 300개 이상의 함수 호출을 이용하여 단순한 기하도형에서부터 복잡한 3차원 장면 생성
  - CAD, 가상현실, 정보시각화, 비행 시뮬레이션, 컴퓨터 게임 등의 분야에서 활용되고 있음
  - Direct3D와 함께 산업계에 널리 사용되고 있음
  - 오픈지엘 사용하여 개발된 대표적인 게임: 퀘이크, 둠3 등

  - OpenGL Architecture Review Board (http://www.opengl.org)
    - 다양한 플랫폼에서 작동되도록 GL을 수정하여 OpenGL 제정
    - 1992년 OpenGL 1.0 발표 이후, 1999년 4월 OpenGL 1.2.1, 2004년 OpenGL 2.0, 2006년에 OpenGL 2.1, 2011년 OpenGL 4.2, 2013년 2월 OpenGL 4.3, 2017년 6월 OpenGL 4.5, 2022년 5월 4.6 출시
    - 비영리 기술 컨소시엄인 Khronos group에서 관리

OpenGL 특징

- 그래픽스 하드웨어에 대한 소프트웨어 인터페이스
  - 하드웨어에 독립적
- 플랫폼에 독립적
  - PC나 워크스테이션 모두에서 가능, 다양한 운영체제 및 호스트 언어를 지원
- 안정성
  - 10년 이상 다양한 플랫폼에서 지원되어 그 사양이 충분히 검증되면서 발전
- 편리성
  - 직관적인 인터페이스와 논리적 명령어들로 구성
- 다양한 그래픽스 기능의 지원으로 응용 소프트웨어 개발이 용이하다.
  - 기본적인 2D 및 3D 그래픽스 함수에서부터 고급 기능까지 지원
  - 셰이딩, 안티알리아싱, 텍스처 매핑, NURBS, 안개, 알파 블렌딩 등

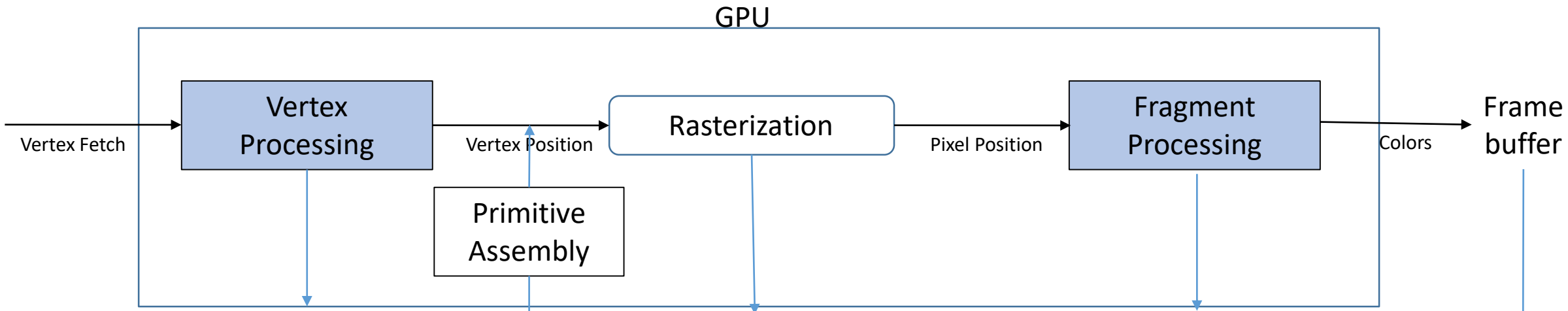
openGL 버전

버전	출시 년도	특징
1.0	1992	IRIS의 GL 버전으로 발표 Fixed-function pipeline 사용
1.2	1998	ARB(Architectural Review Board, 검수 위원회: SGI, Microsoft, Nvidia, HP, 3DLabs, IBM등)이 확장기능 제공
2.0	2005	GLSL 세이더 포함 (fixed-function pipeline도 사용) Vertex shader와 Fragment shader사용
3.0	2008	Deprecation model 소개
3.1	2009.3	Fixed-function pipeline 제거하고 세이더 기반 (Programmable-function pipeline)으로 구현. 모든 응용프로그램은 반드시 vertex shader와 fragment shader를 사용해야 함.
3.2	2009.8	Geometry shader 단계 추가
3.3	2010	핵심 기능 세팅, 이후의 버전에는 방법이 수정이 아닌 추가 GLSL버전이 openGL 버전과 같아짐
4.1	2010.7	Tessellation-control shader와 tessellation-evaluation shader 단계 추가
4.6	2017	벌칸과 상호연동 기능 탑재

- Fixed-function pipeline: 기능이 고정되어 있어 모든 단계는 주어진 연산만 가능
- Programmable-function pipeline: 렌더링 단계의 모든 프로세스를 프로그램에서 설정하여 처리

그래픽스 파이프라인

- 3차원 그래픽스 파이프라인: 3차원 공간에 존재하는 물체를 모니터라는 2차원 평면 위에 보여주기 위한 단계



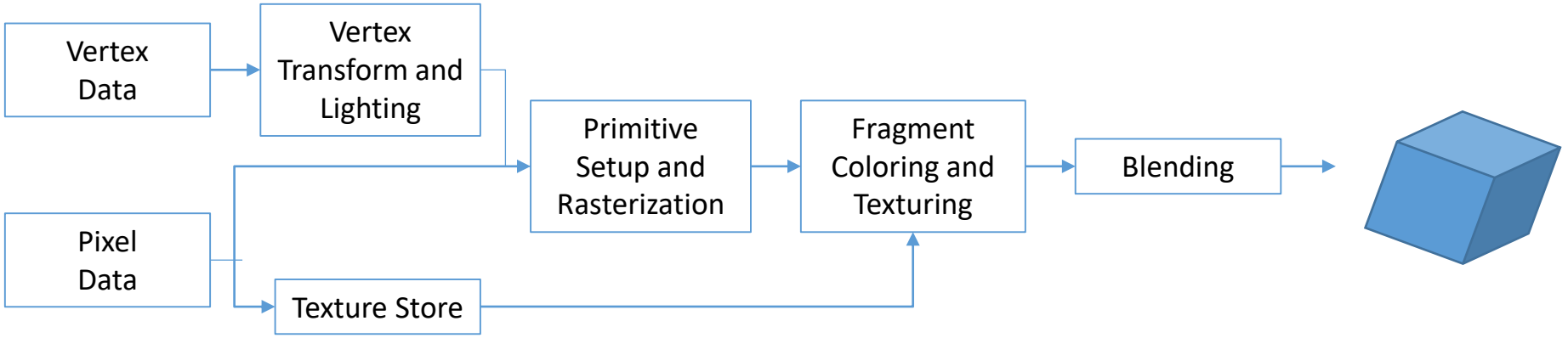
- 정점에 변환, 클리핑, 투영 등을 적용하여 화면 위치로 바꾸기
- 프리미티브들을 화면에 그려질 일련의 프래그먼트들(픽셀)로 변환하는 과정
- 래스터라이저가 생성한 프래그먼트들을 받아들여 픽셀의 색상 계산
- 이후 블렌딩 등 후처리 효과를 계산

- 정점들의 집합을 점, 선, 삼각형 등의 기본 프리미티브 집합으로 변경
- 프리미티브: 컴퓨터 그래픽스 하기 위한 기본적인 도형 요소 (점, 선, 삼각형)

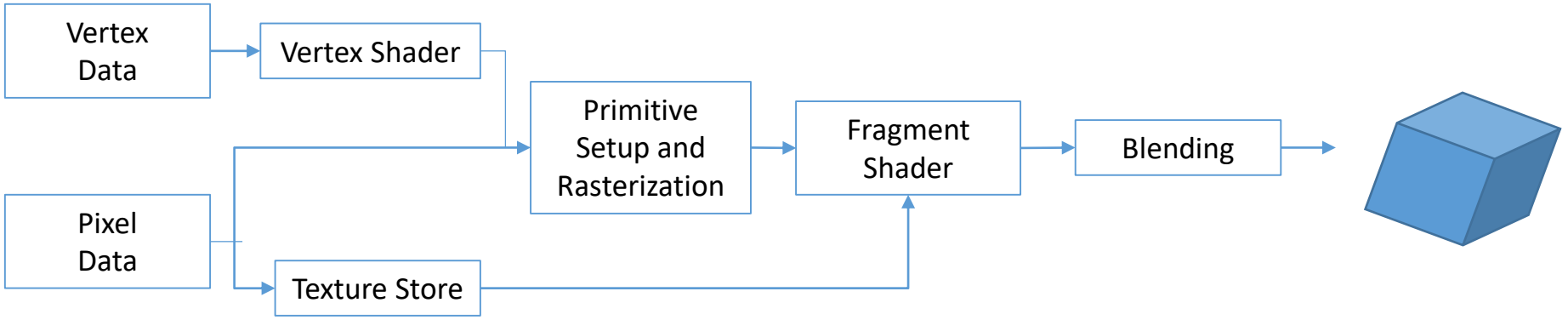
- 화면에 그릴 래스터 이미지
- 그러한 이미지를 생성하는데 필요한 여러 정보를 저장해주는 그래픽스 전용 메모리

그래픽스 파이프라인

- Fixed-function pipeline

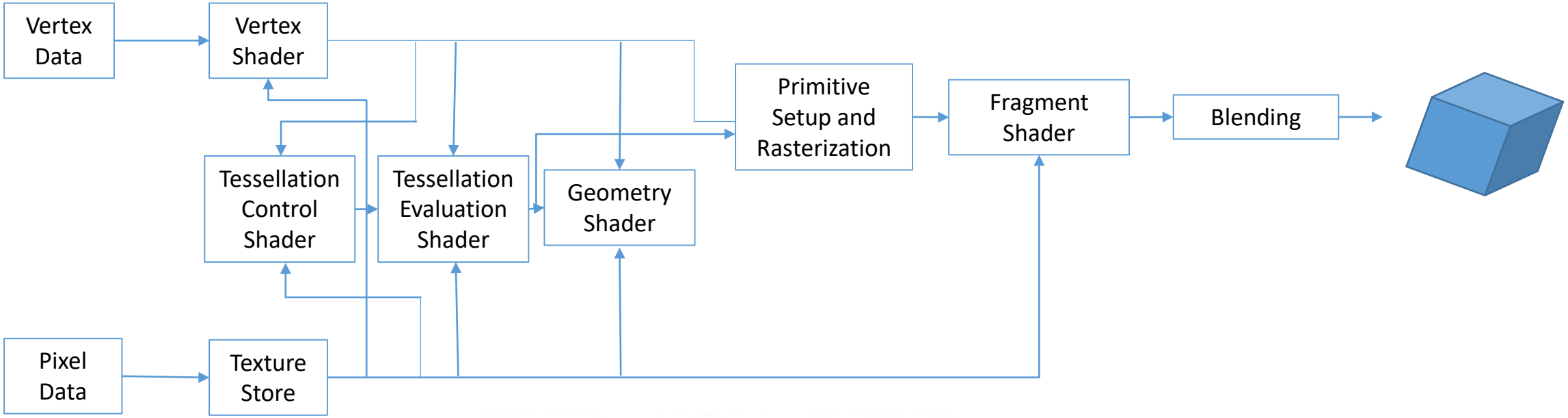


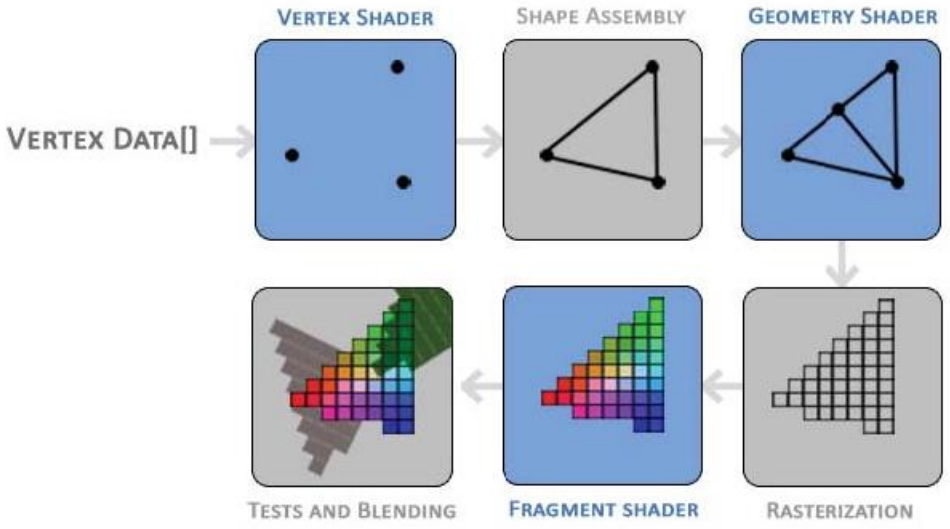
- Programmable-function pipeline



그래픽스 파이프라인

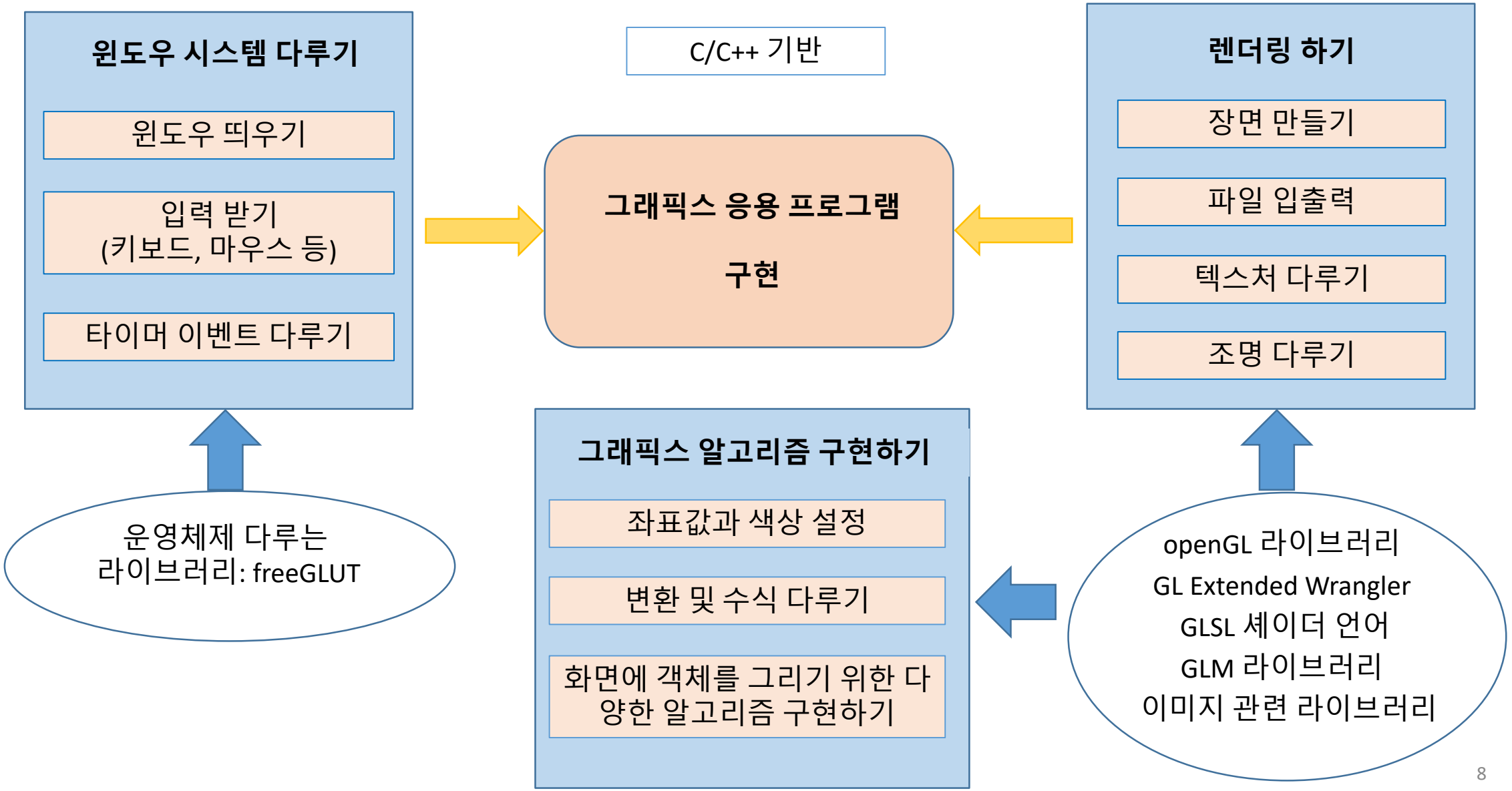
- Programmable-function pipeline





출처: Programmable vs. Fixed-Function Pipeline in Real-Time Computer Graphics
Mladen Korunoski, Marjan Gushev, Vladimir Zdraveski
Published 1 July 2019
Computer Science
IEEE EUROCON 2019 -18th International Conference on Smart Technologies

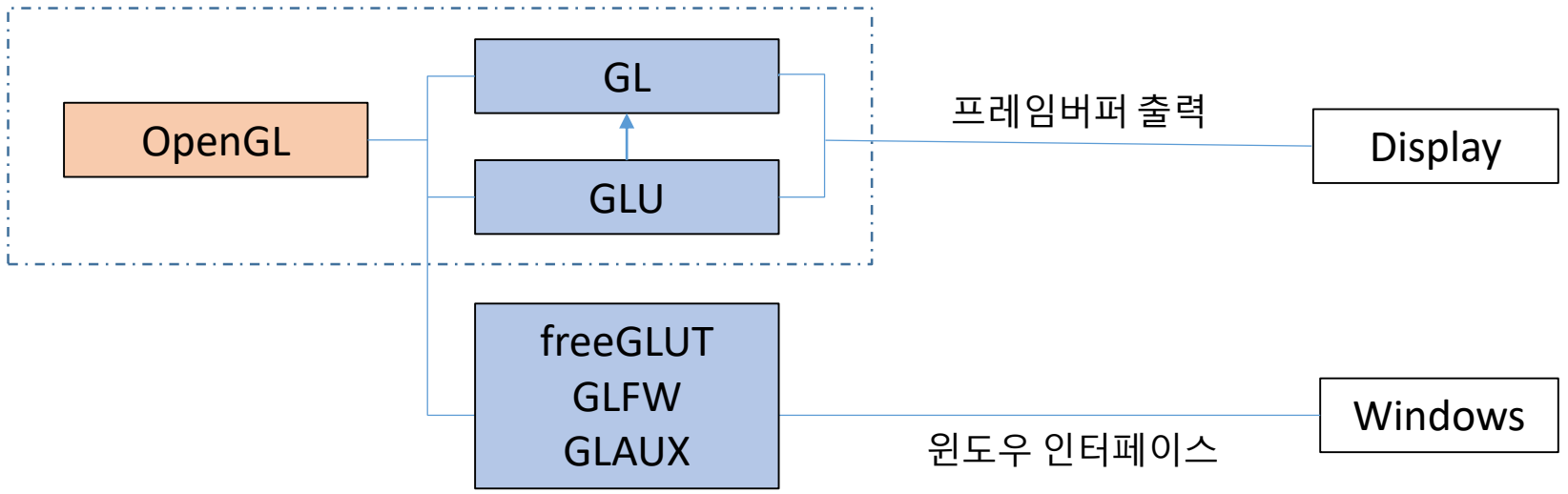
프로그램 구성


윈도우 시스템 다루기
윈도우 띄우기
입력 받기
(키보드, 마우스 등)
타이머 이벤트 다루기
운영체제 다루는
라이브러리: freeGLUT
C/C++ 기반
그래픽스 응용 프로그램
구현
그래픽스 알고리즘 구현하기
좌표값과 색상 설정
변환 및 수식 다루기
화면에 객체를 그리기 위한 다양한 알고리즘 구현하기
렌더링 하기
장면 만들기
파일 입출력
텍스처 다루기
조명 다루기
openGL 라이브러리
GL Extended Wrangler
GLSL 셰이더 언어
GLM 라이브러리
이미지 관련 라이브러리

OpenGL 라이브러리

- 라이브러리 구성
  - GL (openGL Core Library):
    - 렌더링 기능을 제공하는 기본적인 함수들의 집합
  - GLU (openGL Utility Library):
    - GL 함수로 작성되어 있는 고급 기능을 제공하는 함수들의 집합

  - openGL 지원 라이브러리
    - GLUT (openGL Utility Toolkit): 윈도우를 생성, 사용자와의 상호작용 처리 함수들을 지원
    - freeGLUT: 더 이상 개선되지 않는 GLUT의 대체 라이브러리로 윈도우를 다루는 함수들 지원.
    - GLAUX: 오픈지엘 보조 라이브러리
    - GLFW (GL Frame Work): 오픈지엘을 위한 오픈소스 멀티-플랫폼 라이브러리, GLUT와 같은 역할을 하는 라이브러리로 윈도우 생성, 상호작용 처리 함수들을 지원



OpenGL 라이브러리

- 확장 라이브러리:
  - 마이크로소프트 윈도우 운영체제에서 동작하는 비쥬얼 스튜디오는 openGL1.1까지만 지원
  - GLEW (GL Extension Wrangler)라이브러리를 이용하여 openGL2.0 이상의 버전을 사용할 수 있다.
    - 하드웨어와 연관이 되어 있는 openGL은 사용하기 어려움
    - 3.2 버전 이상의 상위 openGL 버전을 사용할 수 있다.
    - GLEW는 GLSL같은 openGL extension을 편리하게 사용할 수 있도록 함
    - openGL 응용 프로그램에서
      - 헤더파일 추가: #include <glew.h>
      - openGL 확장 기능을 사용하기 위해 초기화: glewInit();

      를 불러 초기화 하면 됨.

- 그 외,
  - 셰이더를 위한 openGL Shanding Language (GLSL)
    - openGL의 세이더를 컨트롤할 수 있는 C언어와 유사한 형태의 언어
  - 수학을 위한 GL Mathematics (GLM)
    - GLSL기반으로 하는 그래픽스 소프트웨어에서 사용할 수 있는 C++ 수학 라이브러리
    - 벡터와 행렬 연산을 수행할 때 필요한 함수들 제공

Shader (셰이더)

- GPU (Graphics Process Unit)
  - 그래픽스 연산을 위해 특수하게 제작된 하드웨어
    - 화면 렌더링을 보다 빠르고 효율적으로 처리하기 위해 병렬 연산 코어를 구성
    - 세이더라 불리는 작은 프로그램을 실행하는 세이더 코어로 구성
    - GPU는 수 천개의 코어수를 가지며, 이를 통합하여 많은 양의 일을 수행할 수 있다.
    - 컴퓨터 그래픽스 분야에서 세이더는 소프트웨어 명령의 집합으로 주로 그래픽 하드웨어의 렌더링 효과를 계산하는 데 쓰인다.
      - 렌더링: 컴퓨터 프로그램을 사용하여 모델로부터 영상을 만들어내는 과정

- GLSL은
  - openGL에서 사용할 수 있는 그래픽스 세이딩 언어 (Graphics Library Shading Language)
  - C언어와 흡사한 형태로 Microsoft의 HLSL과 비슷한 세이딩 언어
  - GLSL을 사용하여 세이더 코드를 메인 프로그램과는 분리되게 구현한다.
  - 구현된 코드는 GPU를 통해 렌더링이 수행된다.

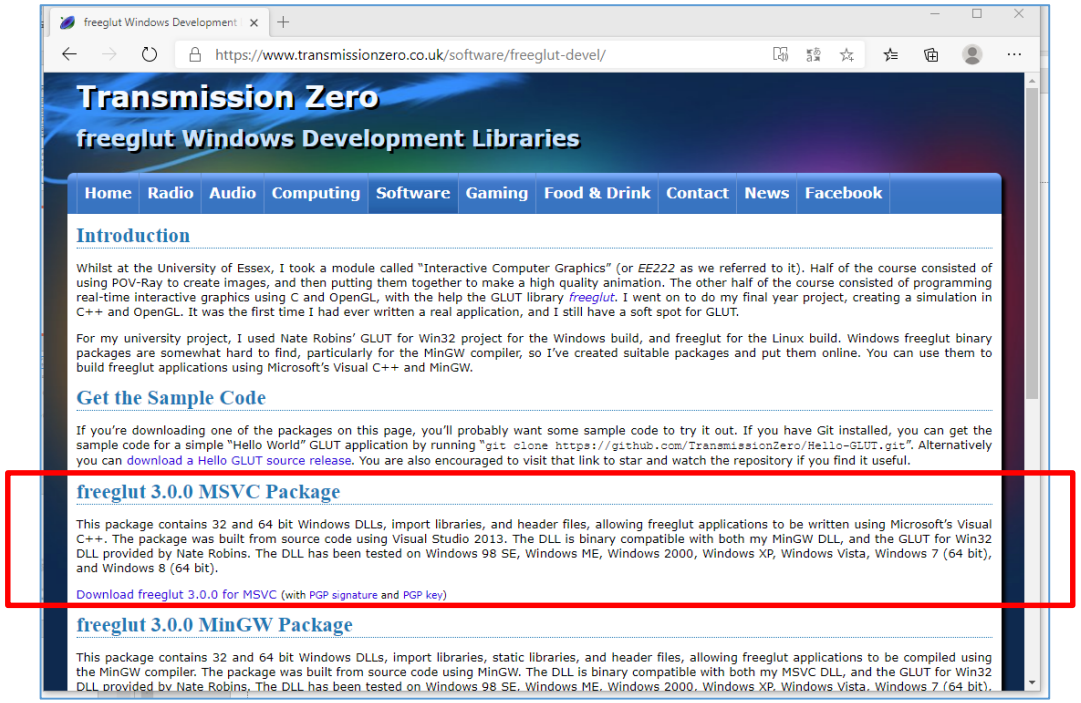
- **세이더 소스 프로그램**은 **메인 프로그램과 분리된 텍스트 파일로 구현**
  - **버텍스 세이더와 프래그먼트 세이더를 각각 따로 구현**
  - 버텍스 세이더와 프래그먼트 세이더로 세이더 프로그램을 생성하여 메인 프로그램에 연결하여 실행

라이브러리 설치: 1. 다운로드 받기

- openGL
  - openGL 표준이 윈도우 시스템에 이미 들어가 있음 (C:\Program Files (x86)\Windows Kits\10 폴더에 저장)

- 라이브러리 다운로드 받기

1) freeGLUT for win32
  - https://www.transmissionzero.co.uk/software/freeglut-devel/ 에서
  - freeglut-MSVC-3.0.0-2.mp.zip 다운로드 받기



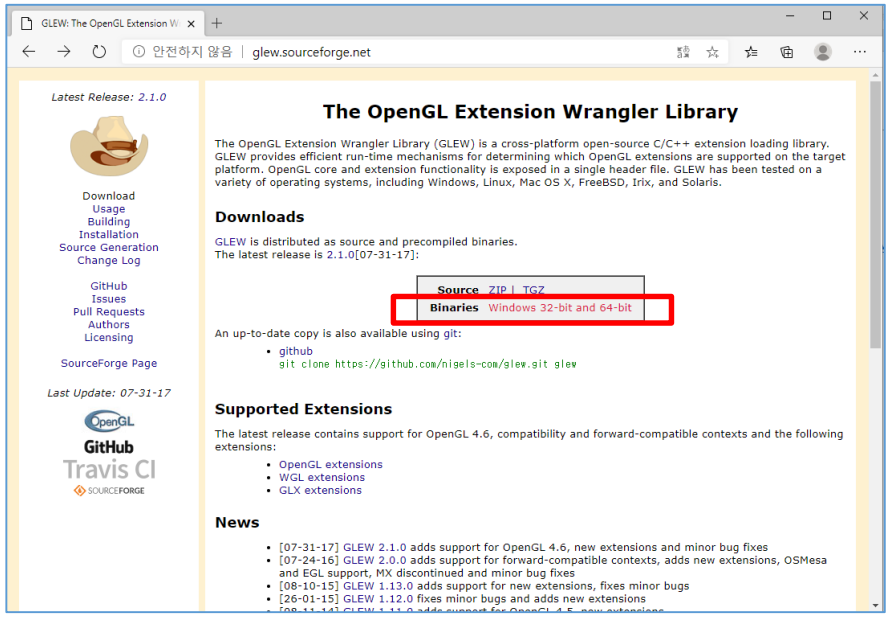
라이브러리 설치: 1. 다운로드 받기

2) GLEW for win32

- http://glew.sourceforge.net 에서
- 바이너리 Windows 32비트 및 64비트에서

  glew-2.0.0-win32.zip 또는 glew-2.1.0-win32.zip 다운로드 받기
  - 우측 그림과 같이 빨간 네모칸의 링크를 누르면 →

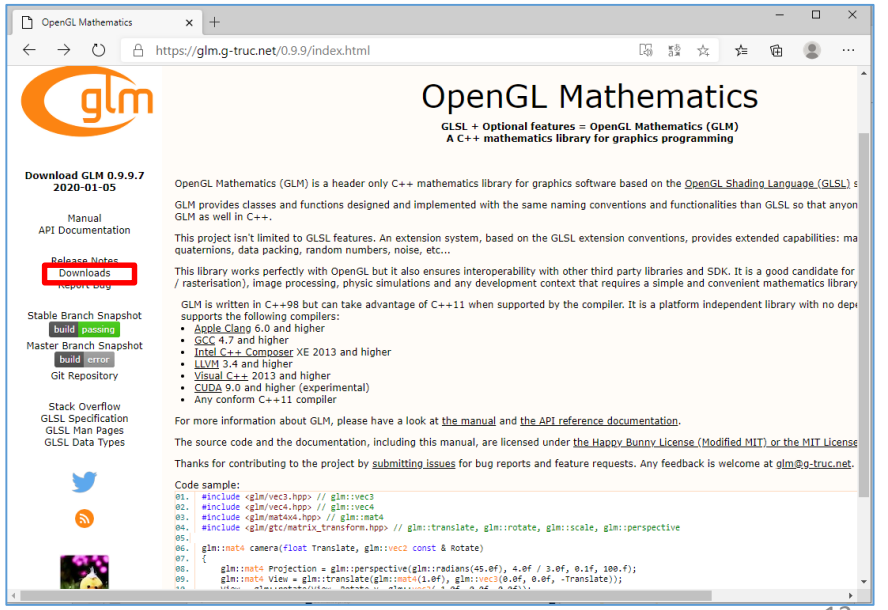
    OpenGL 확장 Wrangler 라이브러리로 이동한 후 다운로드 된다.



3) GLM

- https://glm.g-truc.net/0.9.9/index.html 에서
- glm-0.9.9.5.zip 다운로드 받기
  - 우측 그림과 같이 빨간 네모칸의 Downloads를 누르면 →

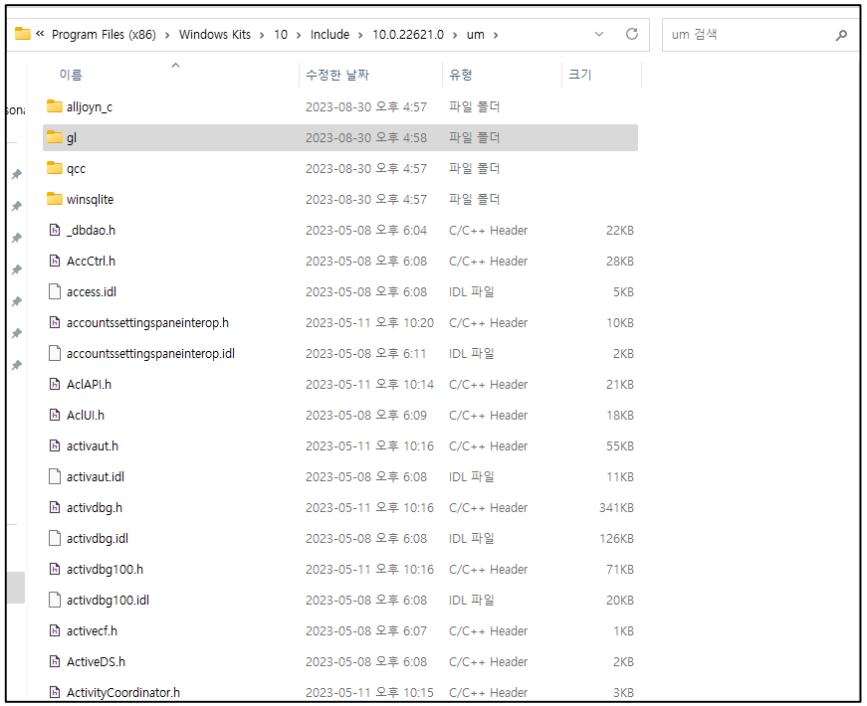
    깃허브로 이동한 후 원하는 버전의 라이브러리를 다운로드한다.
  - 우리는 0.9.9.5 버전을 다운로드 받기로 한다.

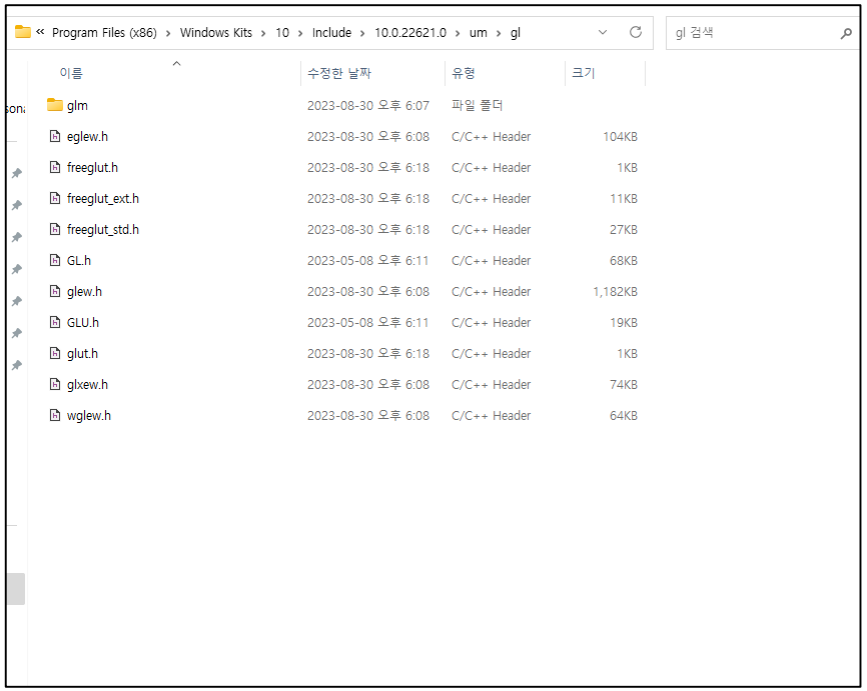


라이브러리 설치: 2. 라이브러리 설치하기

1. include 설치 (헤더 파일 설치)
   - C:\Program Files (x86)\Windows Kits\10\Include\10.0.22621.0\um\gl 폴더에 아래의 h 파일들 저장
     - freeglut 헤더파일들: freeglut.h, freeglut_ext.h, freeglut_std.h
     - GLEW 헤더파일들: glew.h, glxew.h, wglew.h, eglew.h
   - C:\Program Files (x86)\Windows Kits\10\Include\10.0.22621.0\um 폴더에 아래의 폴더 전체 저장 (또는 gl 폴더에)
     - GLM 헤더파일들: glm 폴더에서 glm 디렉토리 전체 (모든 헤더 파일들)

이 숫자는 개인의 pc에 설치된 윈도우의 버전에 따라 다를 수 있음

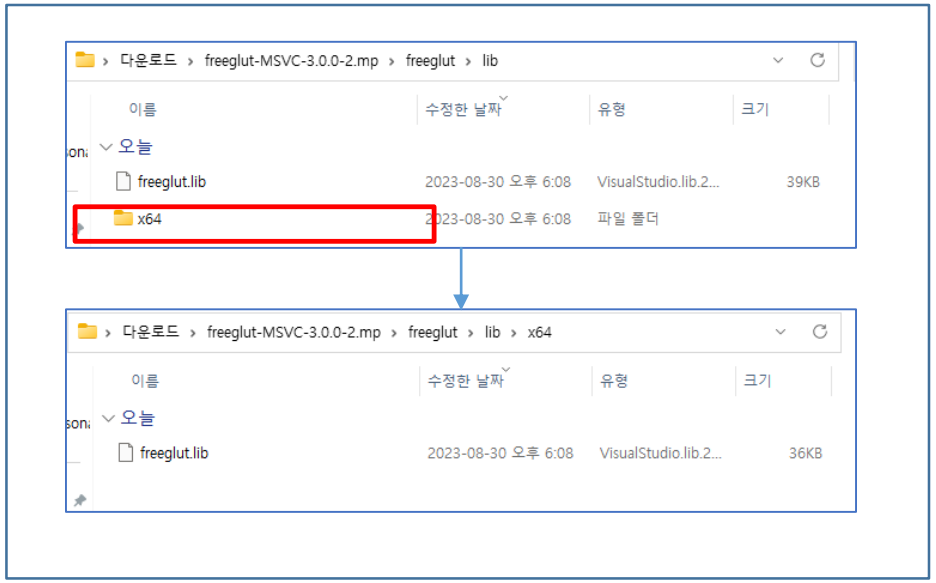


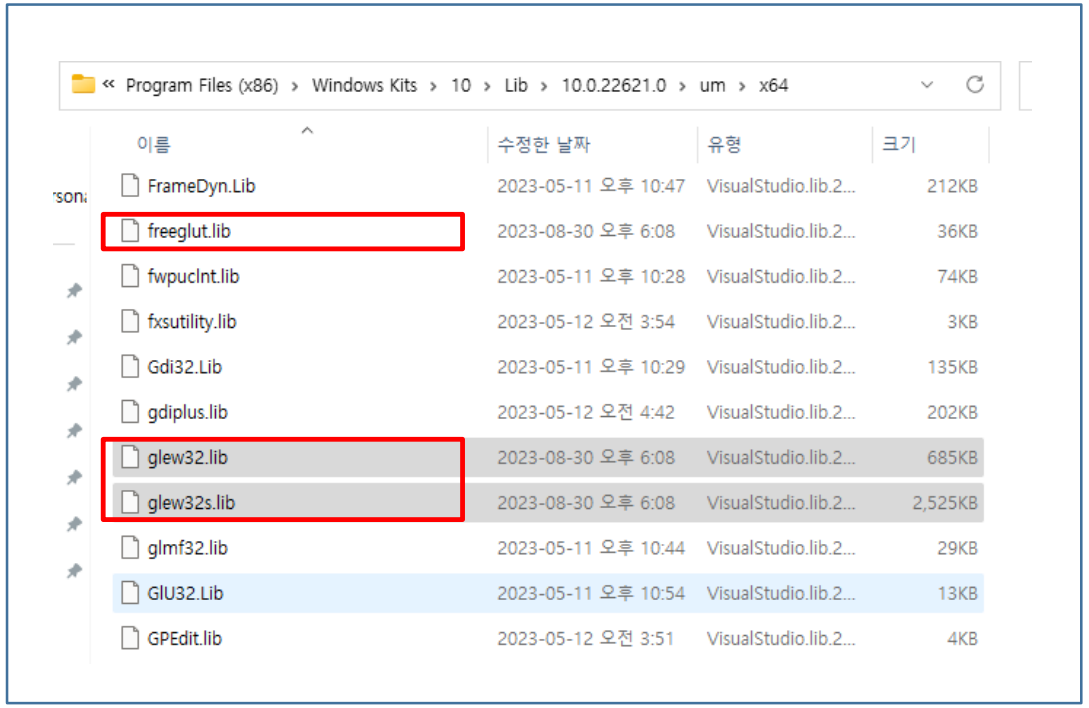


라이브러리 설치: 2. 라이브러리 설치하기

2. lib 설치 (정적 라이브러리 설치)

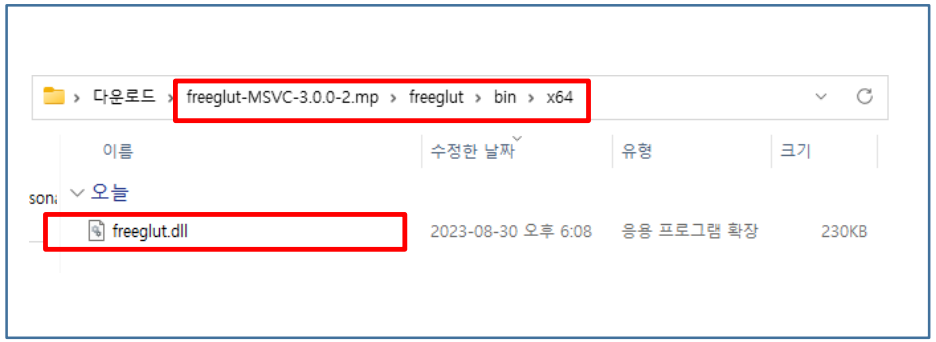
- C:\Program Files (x86)\Windows Kits\10\Lib\10.0.22621.0\um\x64 폴더에 아래의 lib 파일들 저장
  - freeglut 라이브러리: freeglut.lib (..\freeglut-MSVC-3.0.0-2.mp\freeglut\lib\x64 폴더에서 파일 받기)
  - GLEW 라이브러리: glew32.lib 와 glew32s.lib (..\glew-2.1.0-win32\glew-2.1.0\lib\Release\x64)

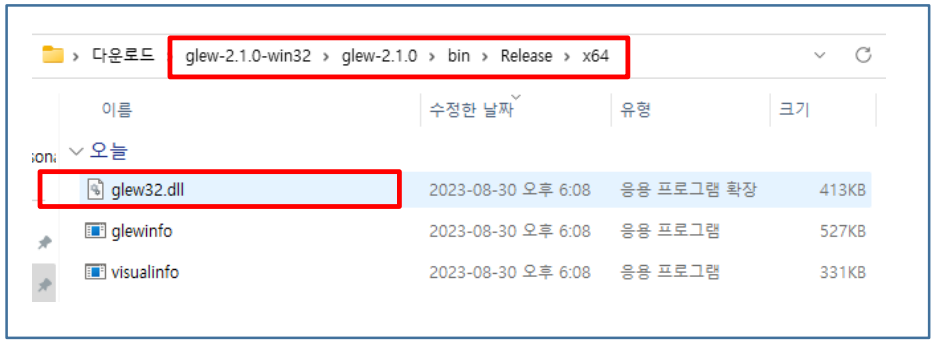


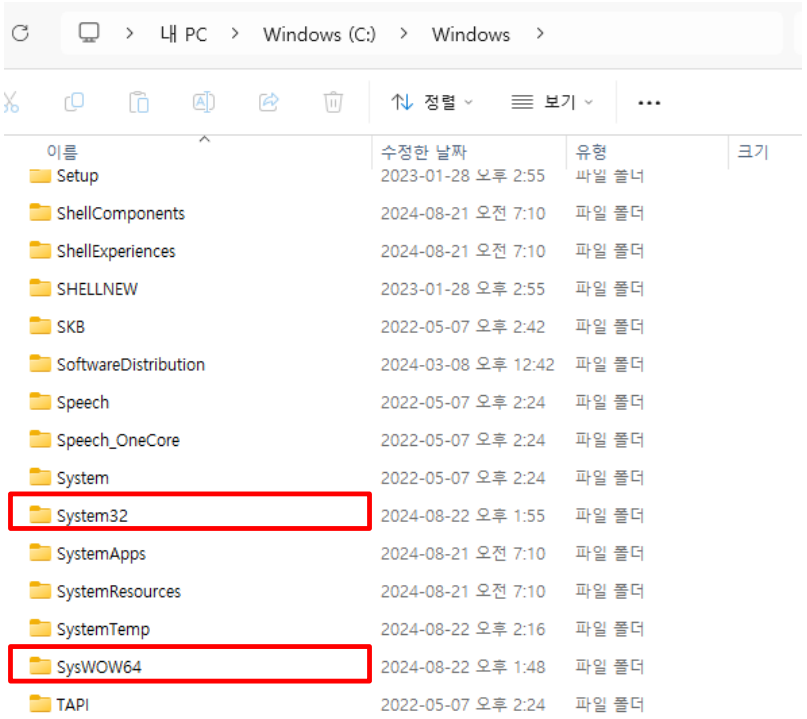


라이브러리 설치: 2. 라이브러리 설치하기

3. dll 설치 (동적 라이브러리 설치)
   - C:\windows\SysWOW64 폴더와 C:\windows\system32 폴더에 아래의 dll 파일들 저장
     - freeglut 라이브러리: freeglut.dll (..\freeglut-MSVC-3.0.0-2.mp\freeglut\bin\x64)
     - GLEW 라이브러리: glew32.dll (..\glew-2.1.0-win32\glew-2.1.0\bin\Release\x64)



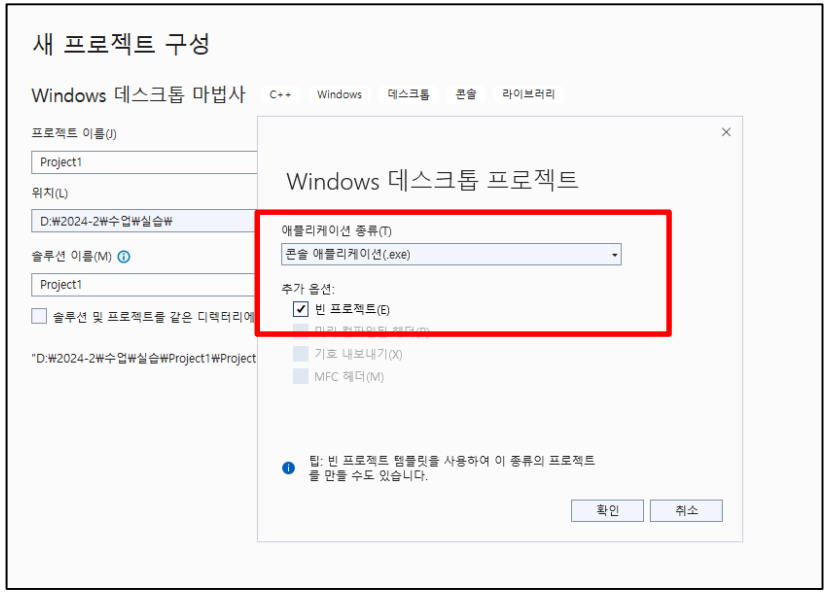


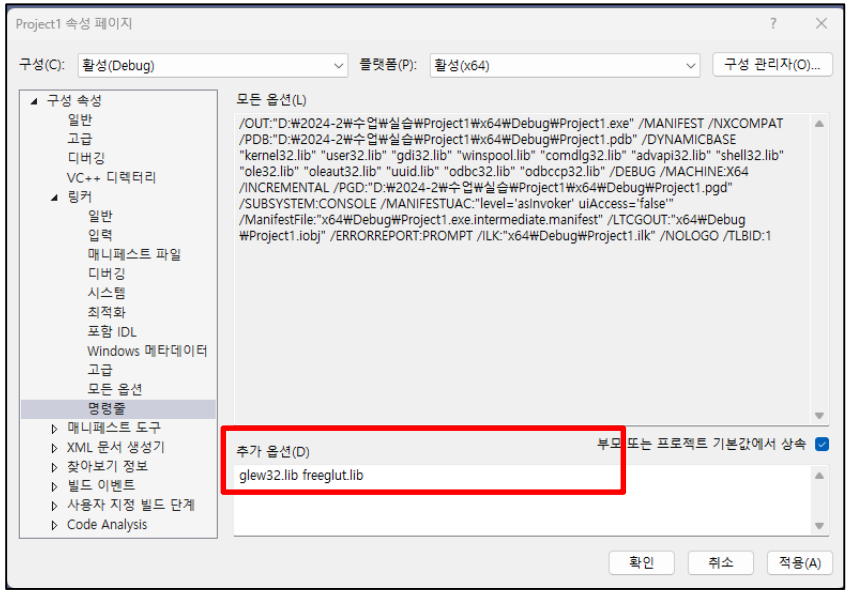


라이브러리 설치 후 프로젝트 만들기

- Visual studio 2022 VC++ 실행
  - 프로젝트 새로 만들기
    - **콘솔 응용 프로그램**으로 프로젝트 만들기
    - **Windows 데스크톱 마법사 → 콘솔 애플리케이션 → 빈 프로젝트**로 프로젝트 만들기

  - 프로젝트 속성 → 링커 → 명령줄에 다음과 같이 2개의 라이브러리 추가
    - **glew32.lib freeglut.lib**





프로젝트 만들기

- Visual studio 2022 VC++ 실행

  - 프로그램에 헤더 파일 넣기

```
#include <gl/glew.h>
#include <gl/freeglut.h>
#include <gl/freeglut_ext.h>
```

* 나중에 좌표계 변환 적용할 때, 아래의 헤더파일 추가로 넣기: glm 헤더파일 추가 (폴더 맞춰서 추가하기)

```
#include <gl/glm/glm.hpp>
#include <gl/glm/ext.hpp>
#include <gl/glm/gtc/matrix_transform.hpp>
```

윈도우 띄우기

```
#include <iostream>
#include <gl/glew.h>
#include <gl/freeglut.h>
#include <gl/freeglut_ext.h>

GLvoid drawScene ( GLvoid );
GLvoid Reshape (int w, int h);

void main ( int argc, char** argv )
{       //--- 윈도우 생성하기
        glutInit (&argc, argv);
        glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGBA );
        glutInitWindowPosition ( 100, 100 );
        glutInitWindowSize ( 250, 250 );
        glutCreateWindow ( "Example1" );

        //--- GLEW 초기화하기
        glewExperimental = GL_TRUE;
        if (glewInit() != GLEW_OK)
        {
                std::cerr << "Unable to initialize GLEW" << std::endl;
                exit(EXIT_FAILURE);
        }
        else
                std::cout << "GLEW Initialized₩n";

        glutDisplayFunc ( drawScene );
        glutReshapeFunc ( Reshape );
        glutMainLoop ();
}

GLvoid drawScene ( )
{
        glClearColor( 0.0f, 0.0f, 1.0f, 1.0f );
        glClear( GL_COLOR_BUFFER_BIT );
        // 그리기 부분 구현: 그리기 관련 부분이 여기에 포함된다.

        glutSwapBuffers ();
}

GLvoid Reshape ( int w, int h )
{
        glViewport ( 0, 0, w, h );
}
```

//--- 필요한 헤더파일 include

//--- 윈도우 출력하고 콜백함수 설정

// glut 초기화
// 디스플레이 모드 설정
// 윈도우의 위치 지정
// 윈도우의 크기 지정
// 윈도우 생성 (윈도우 이름)

// glew 초기화

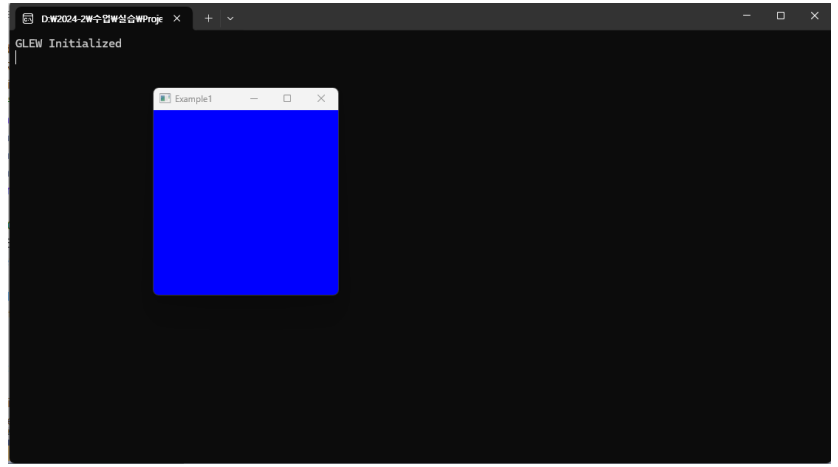
// 출력 함수의 지정
// 다시 그리기 함수 지정
// 이벤트 처리 시작

//--- 콜백 함수: 그리기 콜백 함수

// 바탕색을 'blue' 로 지정
// 설정된 색으로 전체를 칠하기

// 화면에 출력하기

//--- 콜백 함수: 다시 그리기 콜백 함수



기본 데이터 타입

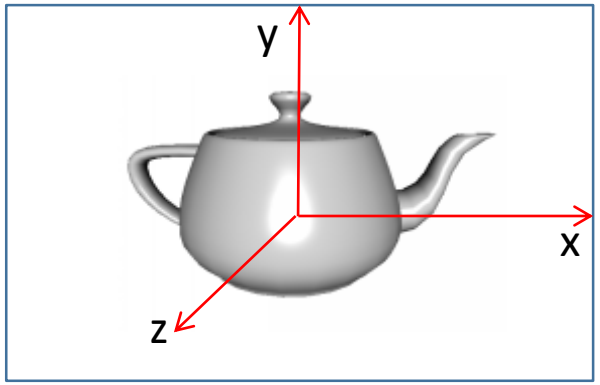
openGL 데이터 형식	표현	C언어 데이터 형식	접미어
GLbyte	8-bit integer	signed char	b
GLshort	16-bit integer	short	s
GLint, GLsizei	32-bit integer	long	i
GLfloat, GLclampf	32-bit floating point	float	f
GLdouble, Glclampd	64-bit floating point	double	d
GLubyte, GLboolean	8-bit unsigned integer	unsigned char	ub
GLushort	16-bit unsigned integer	unsigned short	us
GLuint, GLenum, GLbitfield	32-bit unsigned integer	unsigned long	ui

- 함수 형태
  - 예) **glUniform**{1/2/3/4}{b/s/i/f/d/ub/us/ui}{v}: <접두어><기본 명령><접미어>
    - 접두어: 함수가 속한 라이브러리
    - 기본명령: 함수 내용
    - 접미어: 인자 개수와 데이터 형식 (생략 가능)

    - 실제 함수는, glUniform1f (vLocation, 1.0f), glUniform2 (vLocation, 0, 0), glUniform3f (vLocation, 0.0f, 0.5f, 1.0f)…

좌표 시스템

- 오픈지엘 좌표계 시스템
  - 화면 중앙이 원점으로 설정
  - 오른손 좌표계: 반 시계 방향이 양의 방향
  - 모든 좌표는 3D 공간의 좌표
    - X축 양의 방향: 왼쪽 → 오른쪽
    - Y축 양의 방향: 아래쪽 → 위쪽
    - Z축 양의 방향: 화면 안 → 화면 밖



  - 좌표계의 범위는 각 좌표측이 **[-1.0, 1.0]** 구간 으로 설정되어 있다.
    - 모든 좌표들은 [-1.0, 1.0] 사이에 놓여야 한다.
    - 투영 변환으로 좌표계 범위는 달라질 수 있다.

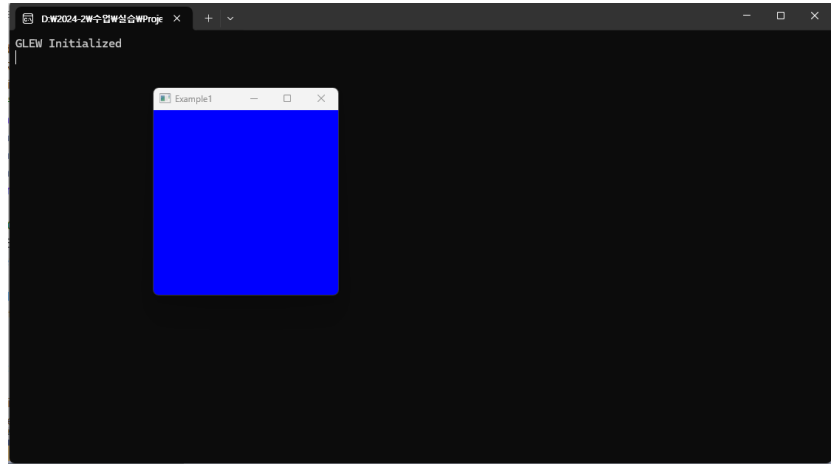
* 오른손 좌표계는 **투영 좌표계를 설정한 이후**부터 적용

- 그 전까지는 카메라가 바라보는 방향, 즉 z축 양의 방향은 화면 안쪽으로 설정되어 있다.

윈도우 띄우기 (19페이지 코드)

```
#include <iostream>
#include <gl/glew.h>
#include <gl/freeglut.h>
#include <gl/freeglut_ext.h>
GLvoid drawScene ( GLvoid );
GLvoid Reshape (int w, int h);

void main ( int argc, char** argv )
{
	//--- 윈도우 생성하기
	glutInit ( &argc, argv );
	glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGBA );
	glutInitWindowPosition ( 100, 100 );
	glutInitWindowSize ( 250, 250 );
	glutCreateWindow ( "Example1" );

	//--- GLEW 초기화하기
	glewExperimental = GL_TRUE;
	if (glewInit() != GLEW_OK)
	{
		std::cerr << "Unable to initialize GLEW" << std::endl;
		exit(EXIT_FAILURE);
	}
	else
		std::cout << "GLEW Initialized₩n";

	glutDisplayFunc ( drawScene );
	glutReshapeFunc ( Reshape );
	glutMainLoop ();
}

GLvoid drawScene ( )
{
	glClearColor( 0.0f, 0.0f, 1.0f, 1.0f );
	glClear( GL_COLOR_BUFFER_BIT );
	// 그리기 부분 구현
	//--- 그리기 관련 부분이 여기에 포함된다.
	glutSwapBuffers ();
}

GLvoid Reshape ( int w, int h )
{
	glViewport ( 0, 0, w, h );
}
```

```
// 필요한 헤더파일 include

//--- 윈도우 출력하고 콜백함수 설정

// glut 초기화
// 디스플레이 모드 설정
// 윈도우의 위치 지정
// 윈도우의 크기 지정
// 윈도우 생성 (윈도우 이름)

// glew 초기화

// 출력 함수의 지정
// 다시 그리기 함수 지정
// 이벤트 처리 시작

//--- 콜백 함수: 그리기 콜백 함수
// 바탕색을 'blue' 로 지정
// 설정된 색으로 전체를 칠하기

// 화면에 출력하기

//--- 콜백 함수: 다시 그리기 콜백 함수
```



윈도우 띄우기

- freeGLUT 라이브러리를 이용하여 윈도우 띄우기
  - glut 초기화
    - void **glutInit** (int *argc, char **argv);
      - GLUT와 openGL 환경 초기화
  - 디스플레이 모드 설정
    - void **glutInitDisplayMode** (unsigned int mode);
      - 컬러모델, 윈도우 버퍼 등 초기의 출력 모드를 결정한다.
      - mode:
        - GLUT_DOUBLE: 더블 버퍼 윈도우 (→ 사용할 것)
        - GLUT_SINGLE: 싱글 버퍼 윈도우 (디폴트 모드)
        - GLUT_RGBA : RGBA 모드 (디폴트 모드)
        - GLUT_DEPTH: 깊이 버퍼 윈도우
      - 1개 이상의 모드인 경우 | 연산자로 연결한다.
  - 윈도우의 위치 지정
    - void **glutInitWindowPosition** (int x, int y);
      - 스크린에서 윈도우의 좌측 상단 모서리에 해당하는 위치를 지정한다.
    - void **glutPositionWindow** (int x, int y);
      - 윈도우의 위치 변화

윈도우 띄우기

- 윈도우의 크기 지정
  - void **glutInitWindowSize** (int width, int height);
    - 윈도우의 크기를 픽셀단위로 지정한다.
    - width, height: 윈도우의 폭과 높이 픽셀 값
  - void **glutReshapeWindow** (int width, int height);
    - 윈도우의 새로운 넓이와 높이

- 윈도우 생성 및 파괴
  - int **glutCreateWindow** (char *string);
    - 윈도우를 생성한다.
    - string: 윈도우 이름
  - void **glutDestroyWindow** (int winID);
    - 윈도우를 파괴한다.
    - win: 윈도우의 id

- Full screen 만들기
  - void **glutFullScreen** (void);
    void **glutLeaveFullScreen** (void);
    void **glutFullScreenToggle** (void);
    - 전체 화면으로 세팅/해제한다.

윈도우 띄우기

- GLEW 초기화
  - 윈도우를 생성한 후 GLEW 초기화
  - openGL extension 정보를 가져올 수 있도록 설정
    - glewExperimental = GL_TRUE;
  - GLEW 라이브러리 초기화
    - glewInit ();
  - GLEW 초기화 성공 후, openGL 라이브러리 함수 호출 가능

  - 사용 예)

```
glewExperimental = GL_TRUE;
if (glewInit() != GLEW_OK)                {                              //-- 초기화 할 수 없을 때 에러 출력
        std::cerr << “Unable to initialize GLEW” << std::endl;
        exit(EXIT_FAILURE);
}
else
        std::cout << “GLEW Initialized\n”;
```

윈도우 띄우기

- GLUT 이벤트 프로세싱 루프 실행
  - void **glutMainLoop** ():
    - 지금까지 생성한 윈도우들과 여기에 그린 그림들을 화면에 출력한다. 또한, 이벤트 처리가 시작되고 디스플레이 콜백으로 등록된 함수가 호출된다.
    - 마우스, 키보드 등의 콜백 함수들이 호출된다.

    - **메인 함수는 최소한 한번의 glutMainLoop 함수를 호출해야 한다. 대개 메인 함수 마무리 부분에서 호출한다.**

  - 이벤트 프로세싱 종료: **glutLeaveMainLoop** ()함수
    - void **glutLeaveMainLoop** **();**
      - 이벤트 프로세싱을 종료 (프로그램 종료)

윈도우 띄우기

- 간단한 openGL 함수들:
  - 화면 클리어 하기
    - void **glClear** (GLbitfield mask)
      - 특정 버퍼나 혼합된 버퍼의 영역을 glClearColor에서 선택한 값으로 설정한다.
        - 컬러 버퍼: GL_COLOR_BUFFER_BIT
        - 깊이 버퍼: GL_DEPTH_BUFFER_BIT
        - 누적 버퍼: GL_ACCUM_BUFFER_BIT
        - 스텐실 버퍼: GL_STENCIL_BUFFER_BIT
    - void **glClearColor** (GLclampf r, GLclampf g, GLclampf b, GLclampf a)
      - 윈도우를 칠할 때의 색 지정
        - r, g, b: red, green, blue 값
        - a: alpha 값 (1.0값으로 고정)

윈도우 띄우기

- 명령어 실행하기
  - void **glFlush** ();
    - openGL 명령을 실행하도록 함
    - 출력 (그리기) 콜백 함수 마지막에 glFlush 함수를 호출하여 (모든 명령어를 실행되게 한다.)
    - **버퍼가 GLUT_SINGLE 인 경우에 사용됨**

- 더블 버퍼링 (double-buffering)
  - void **glutSwapBuffers** ();
    - 그리기를 실행하는 동시에 화면에 나타나지 않는 버퍼(off screen)에 렌더링을 할 수 있다.
    - 스왑(swap) 명령으로 버퍼에 렌더링한 그림을 스크린 상에 즉시 나타낼 수 있다.
    - 더블 버퍼 사용
      - 시간이 오래 걸리는 복잡한 그림을 그린 후 완성된 그림을 화면에 보여줄 수 있다.
      - 애니메이션에서 사용할 수 있다.

  - 사용 방법
    - 메인 함수: 출력 모드를 더블 버퍼링을 위해 설정한다.

      glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB); //--- 출력 모드를 더블 타입으로 설정

    - 그리기 콜백 함수: 드로잉 명령을 실행하고 버퍼 교체를 설정한다.

      glutSwapBuffers (); **//--- 그리기 함수에서 glFlush 대신 사용**
      //--- glFlush 효과가 있으므로 glFlush를 사용할 필요가 없다.
      //--- 그리기 함수 마지막에 호출하여 실행

콜백 함수들

- 콜백 함수
  - 일반 함수 호출: 응용 프로그램이 운영체제에 내장된 함수를 호출하여 원하는 작업을 한다.
  - 콜백 함수: 특정 이벤트나 메시지가 발생했을 때 거꾸로 운영 체제가 이벤트를 처리하기 위하여 응용 프로그램의 함수를 호출하는데 이런 방식으로 불리는 함수를 콜백함수라고 한다.

  - 콜백 함수들:
    - Window 콜백 함수: 출력하기, 다시 그리기, 윈도우 위치/크기 변경, 윈도우에 입력 등 윈도우 관련 콜백 함수
    - Menu 콜백 함수: 메뉴 설정 콜백 함수
    - Global 콜백 함수: 시간 제어와 메뉴 사용 관련 콜백 함수

  - 콜백 함수 등록하려면
    - glut<함수이름>Func 함수를 호출하여 콜백 함수를 등록
      - 예) glutDisplayFunc (…); //--- 출력 콜백 함수 등록

콜백 함수들

함수	함수 설명
glutDisplayFunc	출력 콜백 함수 설정
glutReshapeFunc	다시 그리기 콜백 함수 설정
glutIdleFunc	이벤트가 없을 때 발생하는 콜백 함수 설정
glutTimerFunc	타이머 콜백 함수 설정
glutKeybordFunc	키보드 입력 콜백 함수 설정
glutKeyboardUpFunc	키보드를 뗐을 때 발생하는 콜백 함수 설정
glutSpecialFunc	스페셜 키보드 콜백 함수 설정
glutSpecialUpFunc	스페셜 키보드를 뗐을 때 발생하는 콜백 함수 설정
glutMouseFunc	마우스를 눌렀을 때 발생하는 콜백 함수 설정
glutMotionFunc	마우스를 누른 채로 움직였을 때 발생하는 콜백 함수 설정
glutPassiveMotionFunc	마우스를 누르지 않고 움직였을 때 발생하는 콜백 함수 설정
glutPostRedisplay	현재 화면을 refresh 하는 함수
glutCreateMenu	새로운 pop-up 메뉴 만들기 함수

콜백 함수들

- 출력 함수 지정
  - void **glutDisplayFunc** (void (*func)(void));
    - 현재 윈도우의 출력 콜백 함수 설정
    - 윈도우의 내용을 다시 출력해야 할 필요가 있을 때마다 이 함수로 등록한 콜백 함수를 호출한다. 장면을 다시 그리는데 필요한 루틴들은 모두 이 함수 안에 넣어둔다.

  - 사용 예)

```
void main ( int argc, char *argv[] )
  {
        glutInit (&argc, argv);
        glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGBA );      //--- 디스플레이 모드 설정
        glutInitWindowSize ( 250, 250 );                      //--- 윈도우의 크기 지정
        glutCreateWindow ( "Input Control" );                 //--- 윈도우 생성
        glutDisplayFunc ( drawScene );                        //--- 출력 함수의 지정
        glutMainLoop ();
  }

  void drawScene ()
  {
        glClearColor( 0.0f, 0.0f, 1.0f, 1.0f );               //--- 바탕색을 'blue' 로 지정
        glClear( GL_COLOR_BUFFER_BIT );                       //--- 설정된 색으로 전체를 칠하기
        //--- 그리기 부분 구현

        glutSwapBuffers ();
  }
```

콜백 함수들

- 화면 크기 변했을 때 이벤트 처리 함수 지정
  - void **glutReshapeFunc** (void (*func)(int w, int h));
    - 윈도우 크기가 변경될 때 취할 동작을 지정한다.
    - func: 화면 크기가 변했을 때 호출될 콜백 함수 이름
      - w: 윈도우의 새로운 폭
      - h: 윈도우의 새로운 높이

  - 사용 예)

```
void main ( int argc, char *argv[] )
 {
      glutInit (&argc, argv);
      glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGBA );      // 디스플레이 모드 설정
      glutInitWindowSize ( 250, 250 );                      // 윈도우의 크기 지정
      glutCreateWindow ( "Input Control" );                 // 윈도우 생성
      glutDisplayFunc ( drawScene );                        // 출력 함수의 지정
      glutReshapeFunc (Reshape);                            // 다시 그리기 함수의 지정
      glutMainLoop ();
 }

 void Reshape (int w, int h)
 {
      glViewport (0, 0, w, h);
 }
```

콜백 함수들

- 이벤트가 없을 때 호출되는 함수
  - void **glutIdleFunc** (void (*func))
    - 다른 이벤트가 없을 때 실행
    - 애니메이션 효과를 줄 수 있다.

  - 사용 예)

```
void main ( int argc, char *argv[] )
  {
        glutInit (&argc, argv);
        glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGBA );      // 디스플레이 모드 설정
        glutInitWindowSize ( 250, 250 );                      // 윈도우의 크기 지정
        glutCreateWindow ( "Input Control" );                 // 윈도우 생성
        glutDisplayFunc ( drawScene );                        // 출력 함수의 지정
        glutIdleFunc (idleScene);                             // 아이들 타임에 호출하는 함수의 지정
        glutMainLoop ();
  }

  void idleScene ( )
  {
        x += 0.02;
        y += 0.02;
        glutPostRedisplay ();
  }
```

콜백 함수들

- 키보드 입력
  - void **glutKeyboardFunc** (void (*func)(unsigned char key, int x, int y));
    - 키보드와 인자로 지정한 루틴을 연결하여 키를 누를 때 호출되도록 설정한다.
    - 키보드 입력이 일어날 때마다 ASCII 코드값이 설정된다.
      - key: 입력 키보드,
      - x, y: 키보드 입력할 때의 마우스의 위치

    - 사용 예:

```
void main ( int argc, char *argv[] )
{
        glutInit (&argc, argv);
        glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGBA );        // 디스플레이 모드 설정
        glutInitWindowSize ( 250, 250 );                        // 윈도우의 크기 지정
        glutCreateWindow ( "Input Control" );                   // 윈도우 생성
        glutDisplayFunc ( drawScene );                          // 출력 함수의 지정
        glutKeyboardFunc ( Keyboard );                          // 키보드 입력 콜백 함수
        glutMainLoop ();
}

void Keyboard ( unsigned char key, int x, int y)
{
        switch (key) {
            case 't':       ...; break;
            case 'r':       ...; break;
        }
}
```

콜백 함수들

- ASCII 가 아닌 특수 키인 경우
  - void **glutSpecialFunc** (void (*func)(int key, int x, int y));
    - Key: GLUT_KEY_F1 ~ GLUT_KEY_F12,
      GLUT_KEY_LEFT, GLUT_KEY_RIGHT, GLUT_KEY_UP, GLUT_KEY_DOWN,
      GLUT_KEY_HOME, GLUT_KEY_END, GLUT_KEY_INSERT,
      GLUT_KEY_PAGE_UP, GLUT_KEY_PAGE_DOWN

    - 사용 예:

```
void main ( int argc, char *argv[] )
{
      glutInit (&argc, argv);
      glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGBA );          // 디스플레이 모드 설정
      glutInitWindowSize ( 250, 250 );                          // 윈도우의 크기 지정
      glutCreateWindow ( "Input Control" );                     // 윈도우 생성 (윈도우 이름)
      glutDisplayFunc ( drawScene );                            // 출력 함수의 지정
      glutSpecialFunc (SpecialKeyboard);
      glutMainLoop ();

}

void SpecialKeyboard (int key, int x, int y)
{
      if (key == GLUT_KEY_F1)
            ...;
}
```

콜백 함수들

- CTRL, ALT, SHIFT 키보드 눌려졌는지 확인
  - int **glutGetModifiers** (void);
    - GLUT_ACTIVE_CTRL, GLUT_ACTIVE_ALT, GLUT_ACTIVE_SHIFT 값을 리턴

- 키보드를 뗐을 때 호출되는 콜백 함수 설정
  - void **glutKeyboardUpFunc** (void (*func) (unsigned char key, int x, int y));

콜백 함수들

- 마우스 입력
  - void **glutMouseFunc** (void (*func)(int button, int state, int x, int y));
    - 마우스 버튼과 인자로 지정한 루틴을 연결하여 호출되도록 한다.
      - button (버튼 파라미터): GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON
      - state (상태 파라미터): GLUT_UP, GLUT_DOWN
      - x, y: 윈도우에서 마우스의 위치

    - 사용 예:

```
void main ( int argc, char *argv[] )
{
        glutInit (&argc, argv);
        glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGBA );        // 디스플레이 모드 설
        glutInitWindowSize ( 250, 250 );                        // 윈도우의 크기 지정
        glutCreateWindow ( "Input Control" );                   // 윈도우 생성 (윈도우 이름)
        glutDisplayFunc ( drawScene );                          // 출력 함수의 지정
        glutMouseFunc (Mouse);
        glutMainLoop ();
}

void Mouse (int button, int state, int x, int y)
{
        if ( button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
            std::cout << "x = " << x << " y = " << y << std::endl;
}
```

콜백 함수들

- 마우스 이동 입력
  - void **glutMotionFunc** (void (*func)(int x, int y)):
    - 마우스 버튼을 누른 채 마우스를 움직일 때 호출될 콜백 함수를 등록한다.
  - void **glutPassiveMotionFunc** (void (*func)(int x, int y)):
    - 마우스 버튼을 누르지 않은 채 마우스를 움직일 때 호출될 함수 등록
      - x, y: 마우스의 위치

- 사용 예:

```
bool left_button;

void main ( int argc, char *argv[] )
{
        glutInit (&argc, argv);
        glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGBA );
        glutCreateWindow ( "Input Control" );
        glutDisplayFunc ( drawScene );
        glutMouseFunc ( Mouse );
        glutMotionFunc ( Motion );
        glutMainLoop ();
}
```

```
void Mouse (int button, int state, int x, int y)
{
        if (button == GLUT_LEFT_BUTTON )
                left_button = true;

}

void Motion (int x, int y)
{
        if ( left_button == true )
        {
                ...
        }
}
```

콜백 함수들

- 애니메이션 구현을 위한 타이머 설정 함수
  - void **glutTimerFunc** (unsigned int msecs, (*func)(int value), int value);
    - 타임 아웃이 발생할 경우 호출될 콜백 함수를 등록한다.
      - msecs: 콜백 함수를 호출하기 전까지 기다릴 시간 (밀리세컨 단위)
      - func: 호출할 함수의 이름
      - value: 콜백 함수로 전달할 값
    - 사용 예:

```
void main ( int argc, char *argv[] )
{
        glutInit (&argc, argv);
        glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGBA );
        glutCreateWindow ( "Input Control" );
        glutDisplayFunc ( drawScene );
        glutTimerFunc (100, TimerFunction, 1);          // 타이머 함수 설정
        glutMainLoop ();
}

void TimerFunction (int value)
{
        ...
        glutPostRedisplay ();                           // 화면 재 출력
        glutTimerFunc (100, TimerFunction, 1);          // 타이머함수 재 설정

}
```

- **이 함수는 한 번만 실행되므로 지속적인 애니메이션을 위해서는 타이머 함수 내에 타이머를 다시 호출해야한다.**

콜백 함수들

- 현재 윈도우를 refresh 할 때 호출하는 함수
  - void **glutPostRedisplay** (void)
    - 현재 윈도우를 refresh하게 한다.
    - 출력 자료를 변경한 후 화면 다시 그리기를 하기위하여 그리기 콜백 함수를 호출해야 할 때 불러준다.
    - Refresh 되기 전에 여러 번 호출해도 단 한번만 refresh한다.

- 인자값의 상태 변수로 다양한 값을 읽어올 수 있다.
  - int **glutGet** (int state);
    - State:
      - GLUT_SCREEN_WIDTH: 스크린의 폭 (픽셀 단위)
      - GLUT_SCREEN_HEIGHT: 스크린의 높이 (픽셀 단위)
      - GLUT_ELAPSED_TIME: glutInit이 호출된 이후의 밀리세컨 단위의 시간
      - GLUT_WINDOW_X, GLUT_WINDOW_Y, GLUT_WINDOW_WIDTH, GLUT_WINDOW_HEIGHT…
    - 그 외 여러 값들을 읽어올 수 있다.

메뉴 만들기

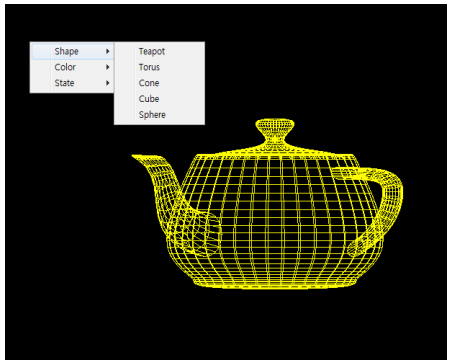
- 팝업 메뉴 만들기
  - 팝업 메뉴를 만든다.
    - int **glutCreateMenu** (void (*func)(int value));
      - 리턴값: 유일한 정수타입의 메뉴 구별자 (1부터 시작한다)
  - 마우스 버튼에 메뉴 삽입하기
    - void **glutAttachMenu** (int button);
    - void **glutDetachMenu** (int button);
      - Button: 버튼 (GLUT_LEFT_BUTTON / GLUT_MIDDLE_BUTTON / GLUT_RIGHT_BOTTON)
  - 메뉴 항목 추가하기
    - void **glutAddMenuEntry** (char *name, int value);
      - Name: 메뉴 엔트리의 이름
      - Value: 메뉴가 선택되면 메뉴의 콜백 함수에 리턴할 값
  - 메뉴의 서브 메뉴 추가하기
    - void **glutAddSubMenu** (char *name, int menu);
      - Name: 서브 메뉴의 이름
      - Menu: 메뉴의 구별자
  - 메뉴 없애기
    - void **glutDestroyMenu** (int menu);

메뉴 만들기

- 사용 예: 오른쪽 마우스를 누르면 3개의 항목이 있는 팝업 메뉴 출력하기

```
void MakeMenu ()
{
        int SubMenu1, SubMenu2, SubMenu3;
        int MainMenu;

        SubMenu1 = glutCreateMenu (MenuFunction);
            glutAddMenuEntry ("Teapot", 1);
            glutAddMenuEntry ("Torus", 2);
            glutAddMenuEntry ("Cone", 3);
            glutAddMenuEntry ("Cube", 4);
            glutAddMenuEntry ("Sphere", 5);

        SubMenu2 = glutCreateMenu (MenuFunction);
            glutAddMenuEntry ("Red", 11);
            glutAddMenuEntry ("Green", 22);

        SubMenu3= glutCreateMenu (MenuFunction);
            glutAddMenuEntry ("Wire", 111);
            glutAddMenuEntry ("Solid", 222);

        MainMenu = glutCreateMenu (MenuFunction);
            glutAddSubMenu ("Shape", SubMenu1);
            glutAddsubMenu ("Color", SubMenu2);
            glutAddsubMenu ("State", SubMenu3);

        glutAttachMenu (GLUT_RIGHT_BUTTON);
}
```

```
void MenuFunction (int button)
{
     //--- button: 메뉴의 구별자값
    switch (button) {
        case 1: ...;      //--- Teapot 인 경우
           break;
        case 2: ...;      //--- Torus 인 경우
           break;
        case 11: ...;     //--- Red 인 경우
           break;
        case 111:...;     //--- Wire인 경우
    }
    glutPostRedisplay ();
}
```



실습 1

- 필요한 라이브러리들을 설치한다.
  - freeGLUT, GLEW, GLM 라이브러리를 설치한다.

- 화면에 800 x 600 크기의 윈도우를 띄운다.
  - 초기 배경색은 흰색
- 윈도우를 띄우고 배경색을 키보드 입력에 따라 다양하게 적용 해 보기
  - 키보드 입력 값:
    - c: 청록색 (초록 + 파랑)
    - m: 자홍색 (빨강 + 파랑)
    - y: 노랑색 (빨강 + 초록)
    - a: 렌덤색
    - w: 백색
    - k: 검정색
    - t: 타이머를 설정하여 특정 시간마다 렌덤색으로 계속 바뀌게 한다.
    - s: 타이머 종료
    - q: 프로그램 종료

실습 1

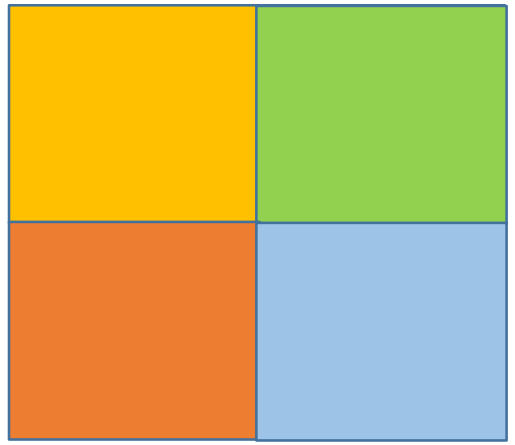
```
#include <iostream>
#include <gl/glew.h>                                        //--- 필요한 헤더파일 include
#include <gl/freeglut.h>
#include <gl/freeglut_ext.h>
GLvoid drawScene ( GLvoid );
GLvoid Reshape (int w, int h);
GLvoid Keyboard ( unsigned char key, int x, int y);


void main ( int argc, char** argv )                         //--- 윈도우 출력하고 콜백함수 설정
{
      //--- 윈도우 생성하기
      glutInit ( &argc, argv );                             //--- glut 초기화
      glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGBA );      //--- 디스플레이 모드 설정
      glutInitWindowPosition ( 0, 0 );                      //--- 윈도우의 위치 지정
      glutInitWindowSize ( 800, 600 );                      //--- 윈도우의 크기 지정
      glutCreateWindow ( "Example1" );                      //--- 윈도우 생성 (윈도우 이름)

      //--- GLEW 초기화하기
      glewExperimental = GL_TRUE;
      if (glewInit() != GLEW_OK)                            //--- glew 초기화
      {
            std::cerr << "Unable to initialize GLEW" << std::endl;
            exit(EXIT_FAILURE);
      }
      else
            std::cout << "GLEW Initialized₩n";

      glutDisplayFunc ( drawScene );                        //--- 출력 콜백함수의 지정
      glutReshapeFunc ( Reshape );                          //--- 다시 그리기 콜백함수 지정
      glutKeyboardFunc ( Keyboard );                        //--- 키보드 입력 콜백함수 지정
      glutMainLoop ();                                      //--- 이벤트 처리 시작
}
```

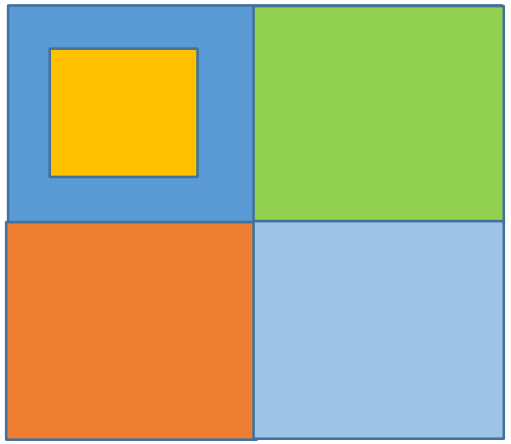
실습 1

```
GLvoid drawScene ( )                                    //--- 콜백 함수: 그리기 콜백 함수
{
      //--- 변경된 배경색 설정
      glClearColor ( ...);                              //--- 바탕색을 변경
      glClear ( GL_COLOR_BUFFER_BIT );                  //--- 설정된 색으로 전체를 칠하기
      glutSwapBuffers ( );                              //--- 화면에 출력하기
}

GLvoid Reshape ( int w, int h )                         //--- 콜백 함수: 다시 그리기 콜백 함수
{
      glViewport ( 0, 0, w, h );
}

GLvoid Keyboard ( unsigned char key, int x, int y)
{
      switch (key) {
      case 'c':          ...; break;                    //--- 배경색을 청록색으로 설정
      case 'm':          ...; break;                    //--- 배경색을 자홍색으로 설정
      case 'y':          ...; break;                    //--- 배경색을 노랑색으로 설정
      ...
      }
      glutPostRedisplay ();                             //--- 배경색이 바뀔 때마다 출력 콜백 함수를 호출하여 화면을 refresh 한다
}
```

실습 2

- 윈도우를 띄우고 마우스 명령을 실행 해 본다.
  - 배경은 본인이 설정한 색으로 그린다.
  - 화면의 가로 세로를 각각 2등분하여 사각형 4개를 그린다.
  - 네 개의 사각형에 마우스를 클릭하여 색상과 크기를 바꾼다.
    - 왼쪽 마우스 클릭
      - 사각형 내부 클릭: 사각형 색상을 랜덤하게 바꾸기
      - 사각형 외부 클릭: 배경색을 랜덤하게 바꾸기
    - 오른쪽 마우스 클릭
      - 사각형 내부 클릭: 사각형 크기 축소 (최소 크기 지정)
      - 사각형 외부 클릭: 사각형 크기 확대, 최대 제한 없이 계속 커진다.



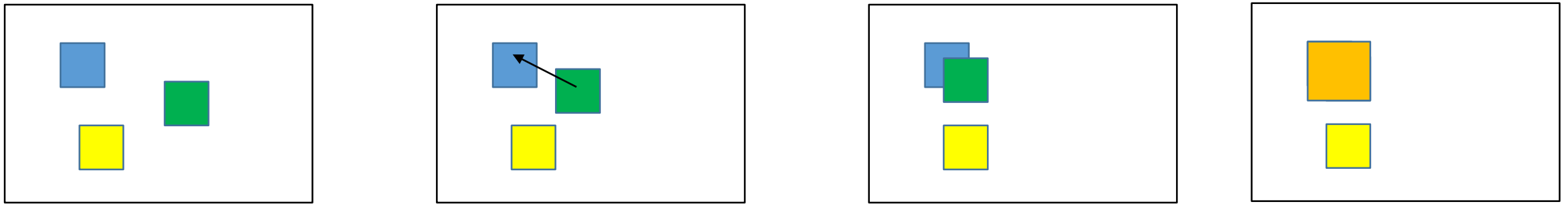


  - 사각형 그리기 함수
    - void glRectf (Glfloat x1, Glfloat y1, Glfloat x2, Glfloat y2);
      - (x1, y1): 좌측 하단 좌표값
      - (x2, y2): 우측 상단 좌표값
  - 사각형 색상 바꾸기 함수
    - void glColor3f (Glfloat r, Glfloat g, Glfloat b);
      - 현재 색상 설정하기
      - (r, g, b): red, green, blue 색상, 0.0 ~ 1.0 사이의 값으로 glRectf 를 호출하기 직전 설정한다.

** Modern openGL 에서는 사용할 수 없는 deprecated 함수
** 셰이더를 사용하게 되면 이 함수는 사용할 수 없음

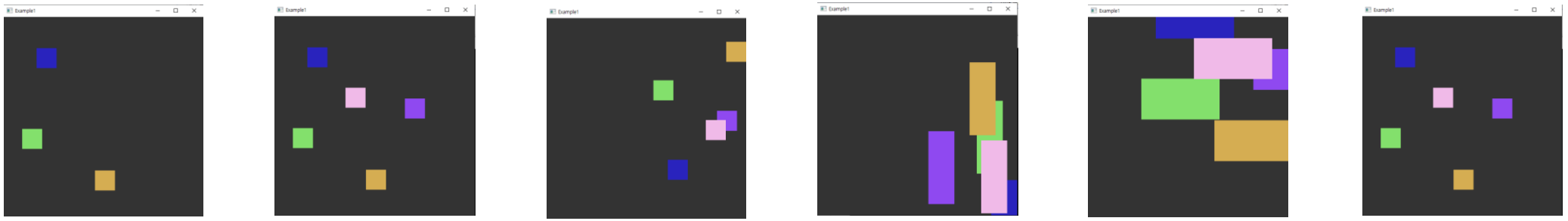
실습 3

- 사각형 이동하기
  - 키보드(키보드 a)를 눌러 화면의 랜덤한 위치에 다른 색상의 사각형을 만든다. 최대 10개를 만든다.
    - 사각형들은 서로 겹쳐져 있을 수 있고, 제일 마지막에 만든 사각형이 맨 위로 올라온다.
  - 마우스 버튼을 사각형 위에 클릭한 채로 드래그 하면
    - 사각형의 위치가 이동된다.
    - 마우스를 놓으면 더 이상 사각형이 이동하지 않는다.
  - 마우스를 이동하여 사각형을 겹치면,
    - 겹쳐진 사각형의 x축 최소, y축 최소값과 x축 최대, y축 최대값으로 큰 사각형으로 만들어진다. 색상은 랜덤하게 바뀐다.



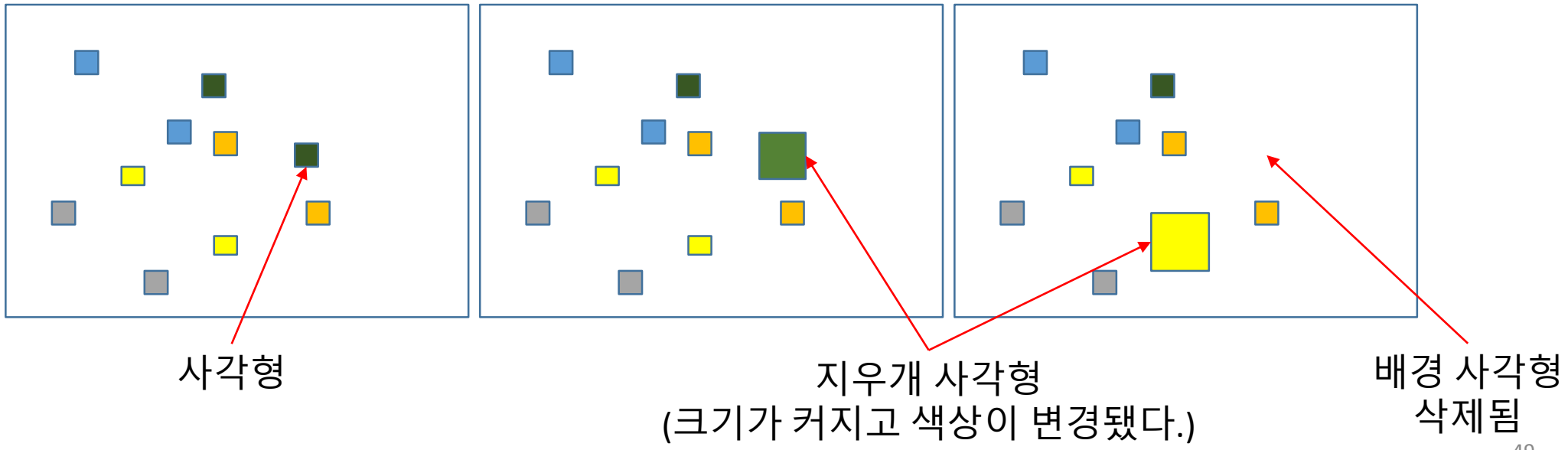
실습 4

- 사각형에 애니메이션 적용하기
  - 배경색은 짙은 회색으로 정하고, 사각형은 랜덤한 색으로 그린다.
  - 마우스를 클릭하는 곳이 중심이 되어 사각형을 그린다. 최대 5개의 사각형을 그린다.
  - 키보드 입력 (명령어는 대문자 또는 소문자 무관함):
    - 1: 위치 변화1 → 사각형들은 각각 대각선으로 이동하고 벽에 닿으면 튕겨 다른 방향으로 이동한다/멈춘다.
    - 2: 위치 변화2 → 사각형들이 지그재그로 이동한다/멈춘다.
    - 3: 크기 변화 → 사각형의 크기가 다양하게 변한다/변하지 않는다.
    - 4: 색상 변화 → 사각형의 색상이 랜덤하게 변한다/변하지 않는다.
    - s: 모든 애니메이션이 멈춘다.
    - m: 원래 그린 위치로 사각형들이 이동한다.
    - r: 사각형들을 삭제하고 다시 마우스 입력을 받을 수 있다.
    - q: 프로그램을 종료한다.



실습 5

- 화면 지우기
  - 윈도우를 띄우고 화면에 같은 크기의 작은 사각형을 다양한 색으로 임의의 위치에 20~40개 그린다.
  - 왼쪽 마우스 버튼을 누르면 화면의 사각형의 2배의 크기의 사각형(지우개 사각형)이 그려지고, 마우스를 누른 채로 이동시키면 지우개 사각형이 위치를 이동한다.
  - 지우개 사각형과 부딪친 사각형은 사라진다. 사각형이 사라지면 지우개 사각형의 크기가 커지고, 지우개 색상은 부딪친 사각형 색상으로 색상을 변경한다.
  - 왼쪽 마우스 버튼을 떼면 지우개 사각형은 사라진다.
    - 다시 마우스를 누르면 검정색의 기존의 지우개 사각형 크기로 지우개 사각형이 생긴다.
  - 키보드 명령어 r: 기존 사각형 삭제되고 새로 그리기



이번주에는

- 오픈지엘 기초
  - 윈도우 다루기
  - 마우스, 키보드 등 입력 방법
  - 애니메이션을 위한 타이머 다루기

- 다음 주에는
  - 셰이더를 위한 GLSL 기초