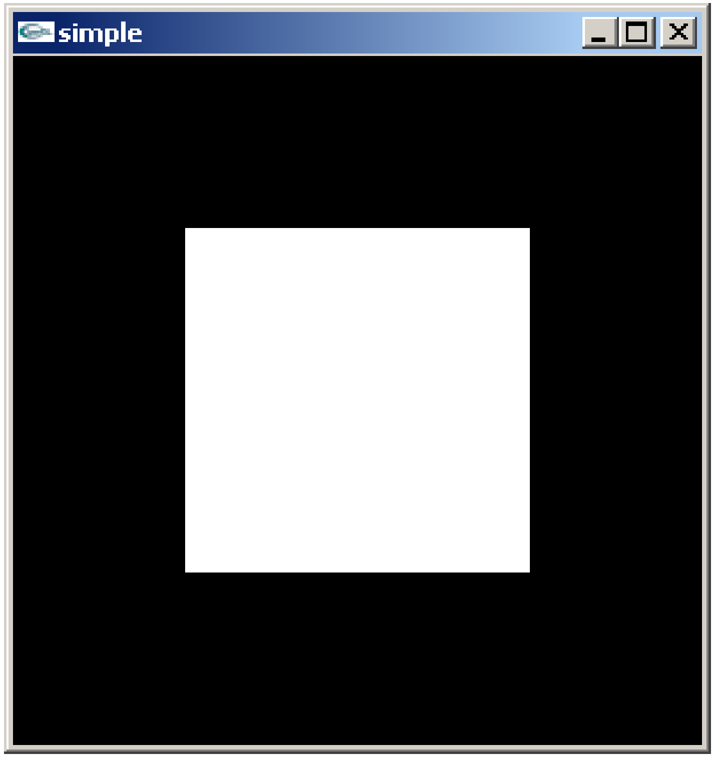
GPU based OpenGL

Sang Il Park

Dept. of Software

A Simple Program (old style)

Generate a square on a solid background



Let’s start to CODE it!

- Preparation

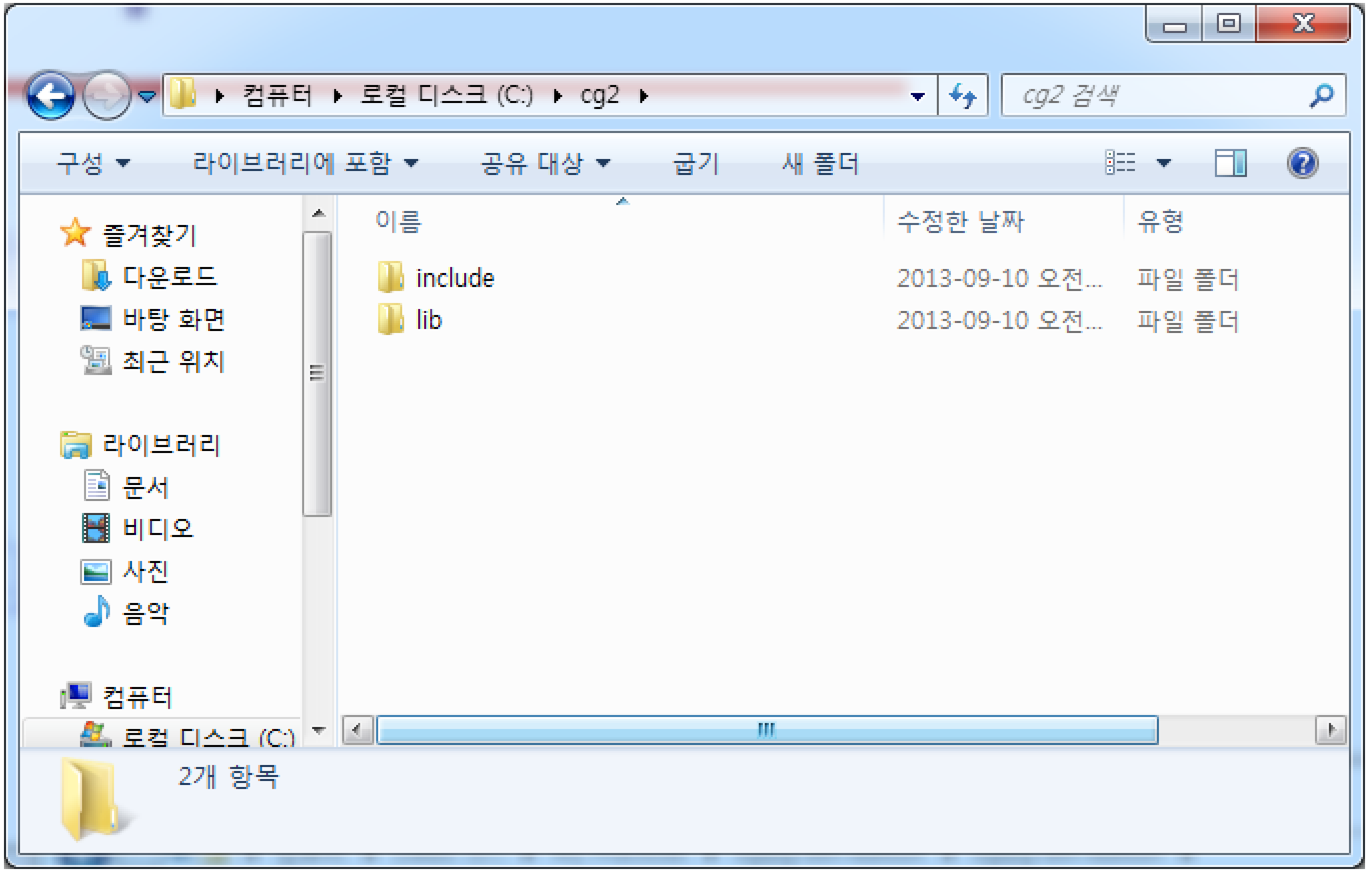
1. Download necessary libraries
   - Header files: Include folder
   - LIB files : lib folder
   - DLL files : bin(or system32) folder
2. Change the project setting
   - Directory setting

Required Libraries

- freeglut: http://freeglut.sourceforge.net/
- GLEW: http://glew.sourceforge.net/

I put both of them and more at our homepage:
freeglut_and_glew.zip
➔ Download it and unzip it at “c:/cg2/”

What you should have in your “c:/cg2/” folder:



▸ 컴퓨터 ▸ 로컬 디스크 (C:) ▸ cg2 ▸

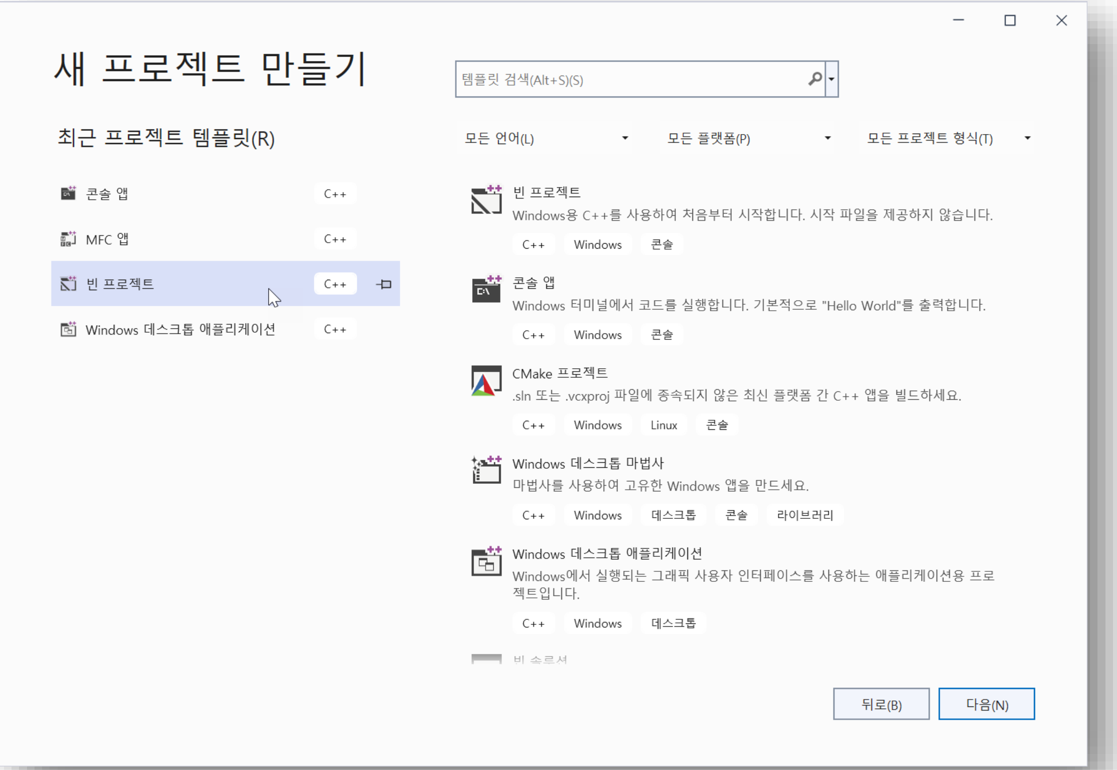
cg2 검색

이름	수정한 날짜	유형
include	2013-09-10 오전...	파일 폴더
lib	2013-09-10 오전...	파일 폴더

2개 항목

Project Setting:

- Start a new project with a "console application" project with an empty project option.



새 프로젝트 만들기

템플릿 검색(Alt+S)(S)

최근 프로젝트 템플릿(R)

- 콘솔 앱 C++
- MFC 앱 C++
- 빈 프로젝트 C++
- Windows 데스크톱 애플리케이션 C++

모든 언어(L) 모든 플랫폼(P) 모든 프로젝트 형식(T)

빈 프로젝트
Windows용 C++를 사용하여 처음부터 시작합니다. 시작 파일을 제공하지 않습니다.
C++ Windows 콘솔

콘솔 앱
Windows 터미널에서 코드를 실행합니다. 기본적으로 "Hello World"를 출력합니다.
C++ Windows 콘솔

CMake 프로젝트
.sln 또는 .vcxproj 파일에 종속되지 않은 최신 플랫폼 간 C++ 앱을 빌드하세요.
C++ Windows Linux 콘솔

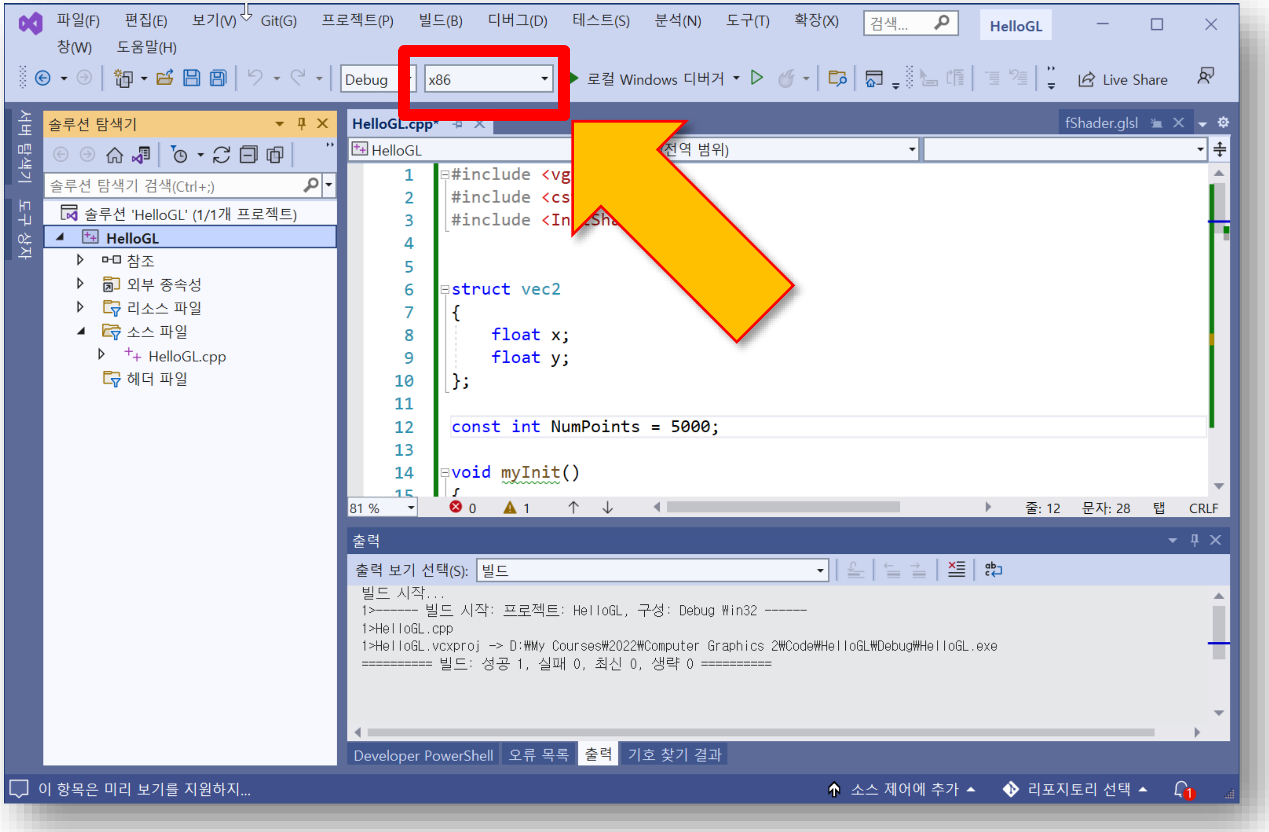
Windows 데스크톱 마법사
마법사를 사용하여 고유한 Windows 앱을 만드세요.
C++ Windows 데스크톱 콘솔 라이브러리

Windows 데스크톱 애플리케이션
Windows에서 실행되는 그래픽 사용자 인터페이스를 사용하는 애플리케이션용 프로젝트입니다.
C++ Windows 데스크톱

빈 솔루션

Set Target Platform: x86

- 다음과 같이 반드시 Target Platform을 x86으로 설정 (x64 아님!)



Debug | x86

```
#include <vg
#include <cs
#include <In  Sha


struct vec2
{
    float x;
    float y;
};

const int NumPoints = 5000;

void myInit()
{
```

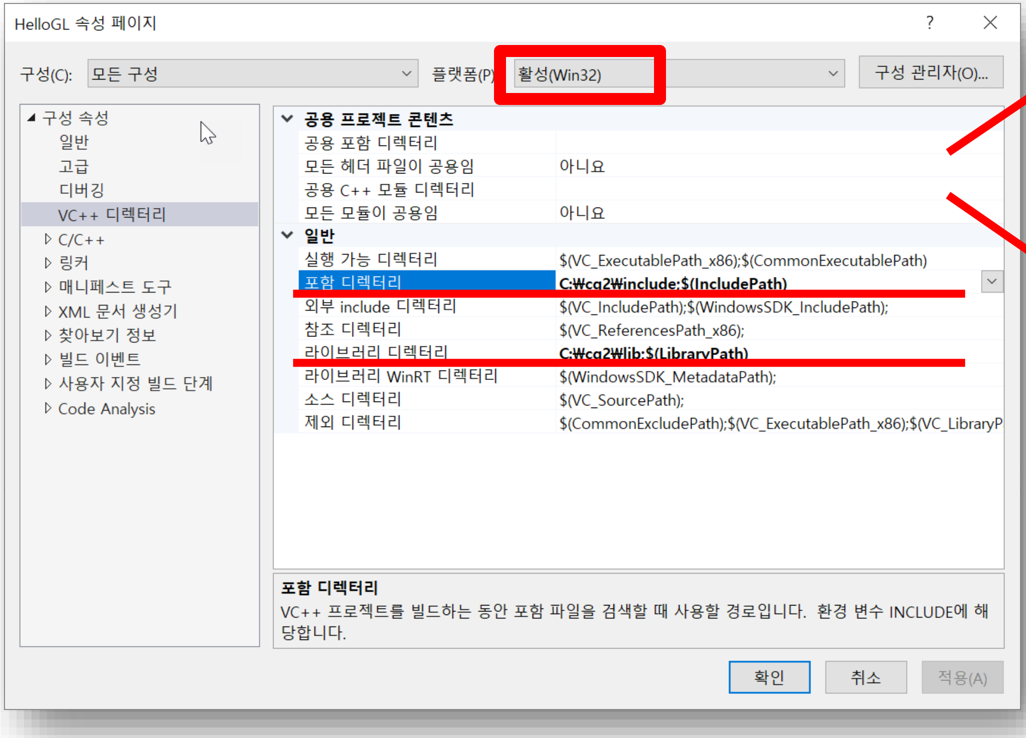
출력

출력 보기 선택(S): 빌드

```
빌드 시작...
1>------ 빌드 시작: 프로젝트: HelloGL, 구성: Debug Win32 ------
1>HelloGL.cpp
1>HelloGL.vcxproj -> D:₩My Courses₩2022₩Computer Graphics 2₩Code₩HelloGL₩Debug₩HelloGL.exe
========== 빌드: 성공 1, 실패 0, 최신 0, 생략 0 ==========
```

Project Setting

- Set the directories:

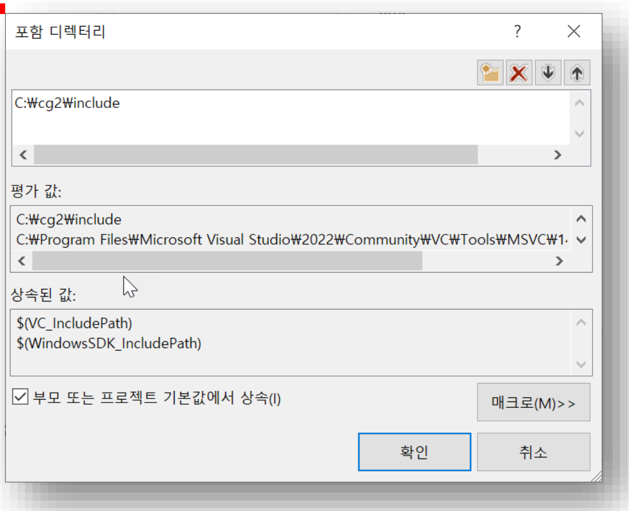


HelloGL 속성 페이지

구성(C): 모든 구성 플랫폼(P): 활성(Win32) 구성 관리자(O)...

공용 프로젝트 콘텐츠	
공용 포함 디렉터리	
모든 헤더 파일이 공용임	아니요
공용 C++ 모듈 디렉터리	
모든 모듈이 공용임	아니요
일반	
실행 가능 디렉터리	$(VC_ExecutablePath_x86);$(CommonExecutablePath)
포함 디렉터리	**C:₩cg2₩include;$(IncludePath)**
외부 include 디렉터리	$(VC_IncludePath);$(WindowsSDK_IncludePath);
참조 디렉터리	$(VC_ReferencesPath_x86);
라이브러리 디렉터리	**C:₩cg2₩lib;$(LibraryPath)**
라이브러리 WinRT 디렉터리	$(WindowsSDK_MetadataPath);
소스 디렉터리	$(VC_SourcePath);
제외 디렉터리	$(CommonExcludePath);$(VC_ExecutablePath_x86);$(VC_LibraryP

포함 디렉터리
VC++ 프로젝트를 빌드하는 동안 포함 파일을 검색할 때 사용할 경로입니다. 환경 변수 INCLUDE에 해당합니다.



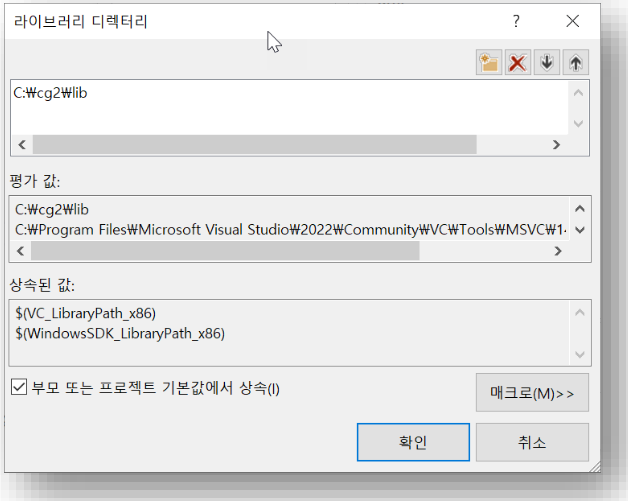
포함 디렉터리

C:₩cg2₩include

평가 값:
C:₩cg2₩include
C:₩Program Files₩Microsoft Visual Studio₩2022₩Community₩VC₩Tools₩MSVC₩1.

상속된 값:
$(VC_IncludePath)
$(WindowsSDK_IncludePath)

☑ 부모 또는 프로젝트 기본값에서 상속(I)



라이브러리 디렉터리

C:₩cg2₩lib

평가 값:
C:₩cg2₩lib
C:₩Program Files₩Microsoft Visual Studio₩2022₩Community₩VC₩Tools₩MSVC₩1.

상속된 값:
$(VC_LibraryPath_x86)
$(WindowsSDK_LibraryPath_x86)

☑ 부모 또는 프로젝트 기본값에서 상속(I)

Create a new main.cpp file

- And add the following line at the beginning of the code:

```
#include <vgl.h>
```

Now ALL SET!!!!

Hello GL Program:

```
#include <vgl.h>

void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_TRIANGLES);
                glVertex2f(-0.5, -0.5);
                glVertex2f(0.5, -0.5);
                glVertex2f(-0.5, 0.5);
                glVertex2f(0.5, -0.5);
                glVertex2f(0.5, 0.5);
                glVertex2f(-0.5, 0.5);
        glEnd();
        glFlush();
}
```

```
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
        glutInitWindowSize(512, 512);
        glutCreateWindow("Hello GL");

        glutDisplayFunc(display);
        glutMainLoop();

        return 0;
}
```

Summary

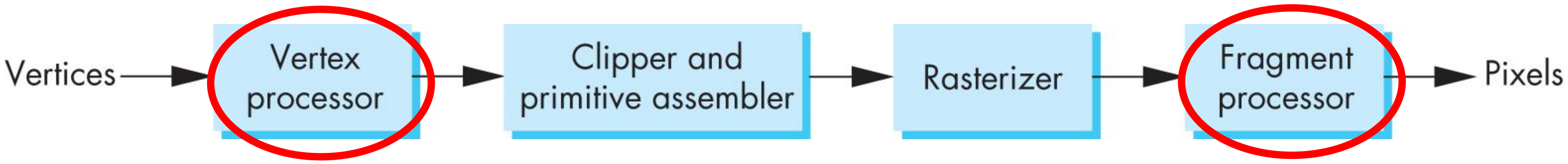
- OpenGL Setting for libraries
  - Set include/lib folder
  - `#include <vgl.h>`

- With OpenGL, drawing was done by just sending the data into the predefined pipeline

Programming with OpenGL in a modern way

Changing in OpenGL

- Performance is achieved by using GPU rather than CPU
- Control GPU through programs called ***shaders***



“new” OpenGL

- We’ll concentrate on the latest versions of OpenGL
  - The currently available version is 4.5
  - At least higher than OpenGL 3.0 will work fine with the class

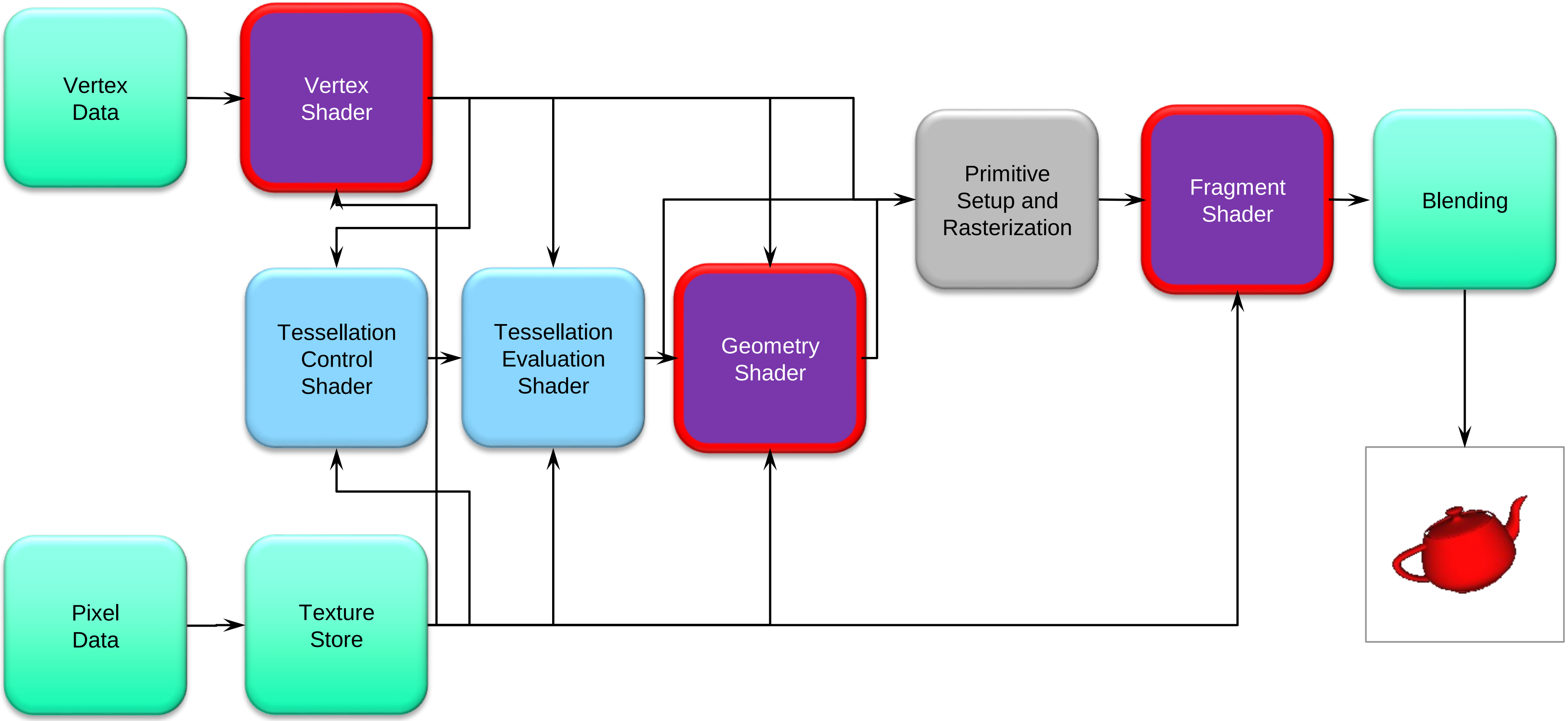
- They enforce a new way to program with OpenGL
  - Allows more efficient use of GPU resources

- modern OpenGL doesn’t support
  - Fixed-function graphics operations
    - lighting
    - transformations

- All applications must use ***shaders*** for their graphics processing

The Latest Pipelines

- Latest version is 4.5 (2014)


Vertex Data
Vertex Shader
Tessellation Control Shader
Tessellation Evaluation Shader
Geometry Shader
Primitive Setup and Rasterization
Fragment Shader
Blending
Pixel Data
Texture Store

OpenGL Libraries

- OpenGL core library
  - OpenGL32 on Windows
  - GL on most unix/linux systems (libGL.a)
- OpenGL Utility Library (GLU)
  - Provides functionality in OpenGL core but avoids having to rewrite code
  - Will only work with legacy code
- Links with window system
  - GLX for X window systems
  - WGL for Windows
  - AGL for Macintosh

GLUT

- OpenGL Utility Toolkit (GLUT)
  - Provides functionality common to all window systems
    - Open a window
    - Get input from mouse and keyboard
    - Menus
    - Event-driven
  - Code is portable but GLUT lacks the functionality of a good toolkit for a specific platform
    - No slide bars

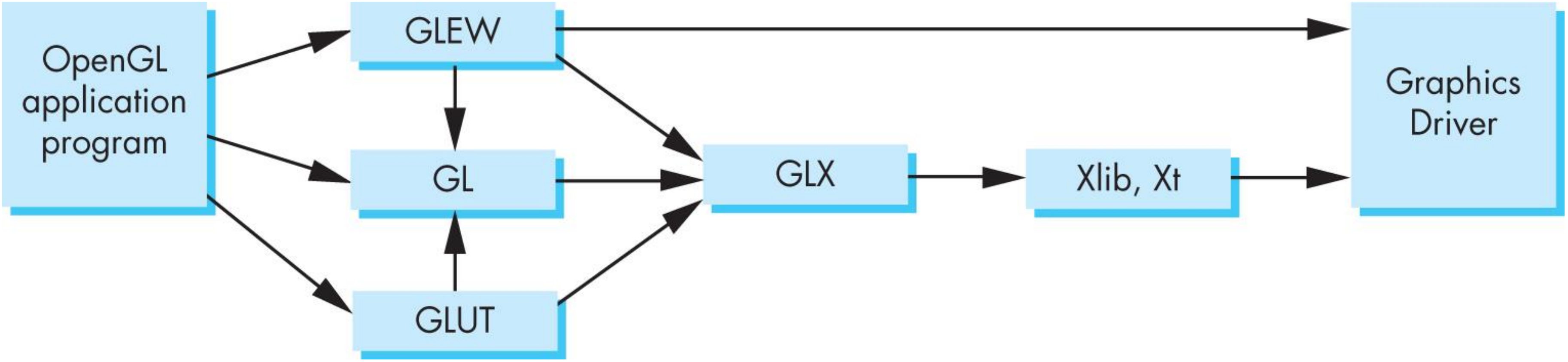
freeGLUT

- GLUT was created long ago and has been unchanged
  - Amazing that it works with OpenGL 3.1
  - Some functionality can’t work since it requires deprecated functions
- *freeglut* updates GLUT
  - Added capabilities
  - Context checking

GLEW

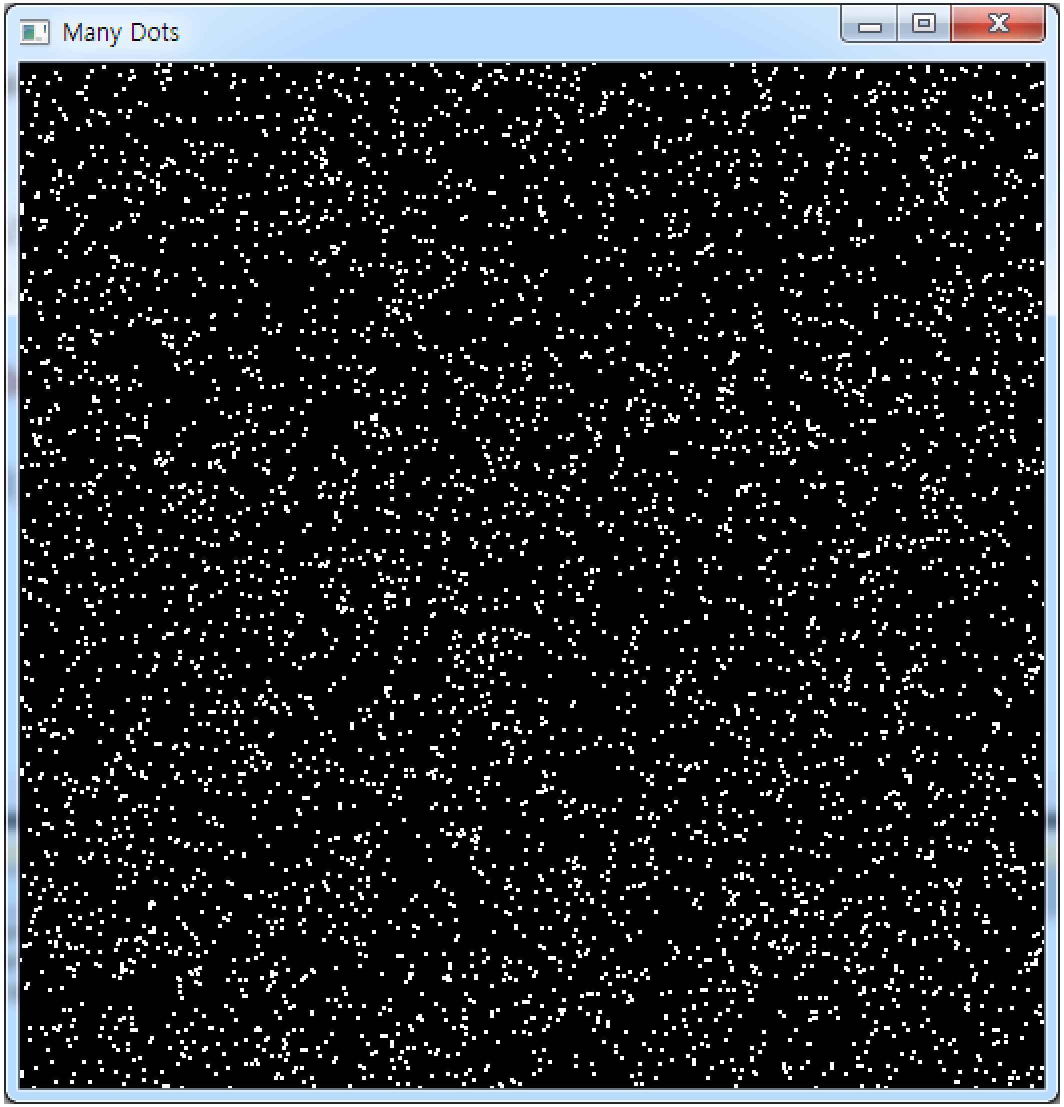
- OpenGL Extension Wrangler Library
- Makes it easy to access OpenGL extensions available on a particular system
- Avoids having to have specific entry points in Windows code
- Application needs only to include glew.h and run a glewInit()

Software Organization



Three different drawing approaches: Many points

Drawing random dots on the screen



Three approaches

1. Immediate mode graphics
2. Retained mode graphics
3. GPU based graphics

Immediate mode graphics

- Generate one primitive at a time and draw it immediately

```
void display()
{
     for(num_points)
     {
          p = generate_a_point();
          Draw_a_point(p);
     }
}
```

Immediate mode graphics:

```
#include <vgl.h>

void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POINTS);
        for(int i=0; i<5000; i++)
        {
                float x = (rand()%200)/100.0f-1.0f;
                float y = (rand()%200)/100.0f-1.0f;
                glVertex2f(x,y);
        }
        glEnd();
        glFlush();
}
```

```
int main(int argc, char ** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
        glutInitWindowSize( 512, 512 );
        glutCreateWindow("Many Points");

        glutDisplayFunc(display);
        glutMainLoop();

        return 0;
}
```

Retained mode Graphics

- Generate all primitives first, then draw them all

```
p[num_points];
void initialize()
{
       for(num_points)
       {
              q = generate_a_point();
              p = store_the_point(q);
       }
}

void display()
{
       draw_all_the_points(p)
}
```

GPU-based Graphics

- Generate all the points first, send them to GPU, and then draw them.

```
p[num_points];
void initialize()
{
       for(num_points)
       {
              q = generate_a_point();
              p = store_the_point(q);
       }
       Send_all_points_to_GPU(p);
}

void display()
{
       display_data_on_GPU();
}
```

What is different from the previous one?

A First Program:
Many points
with GPU

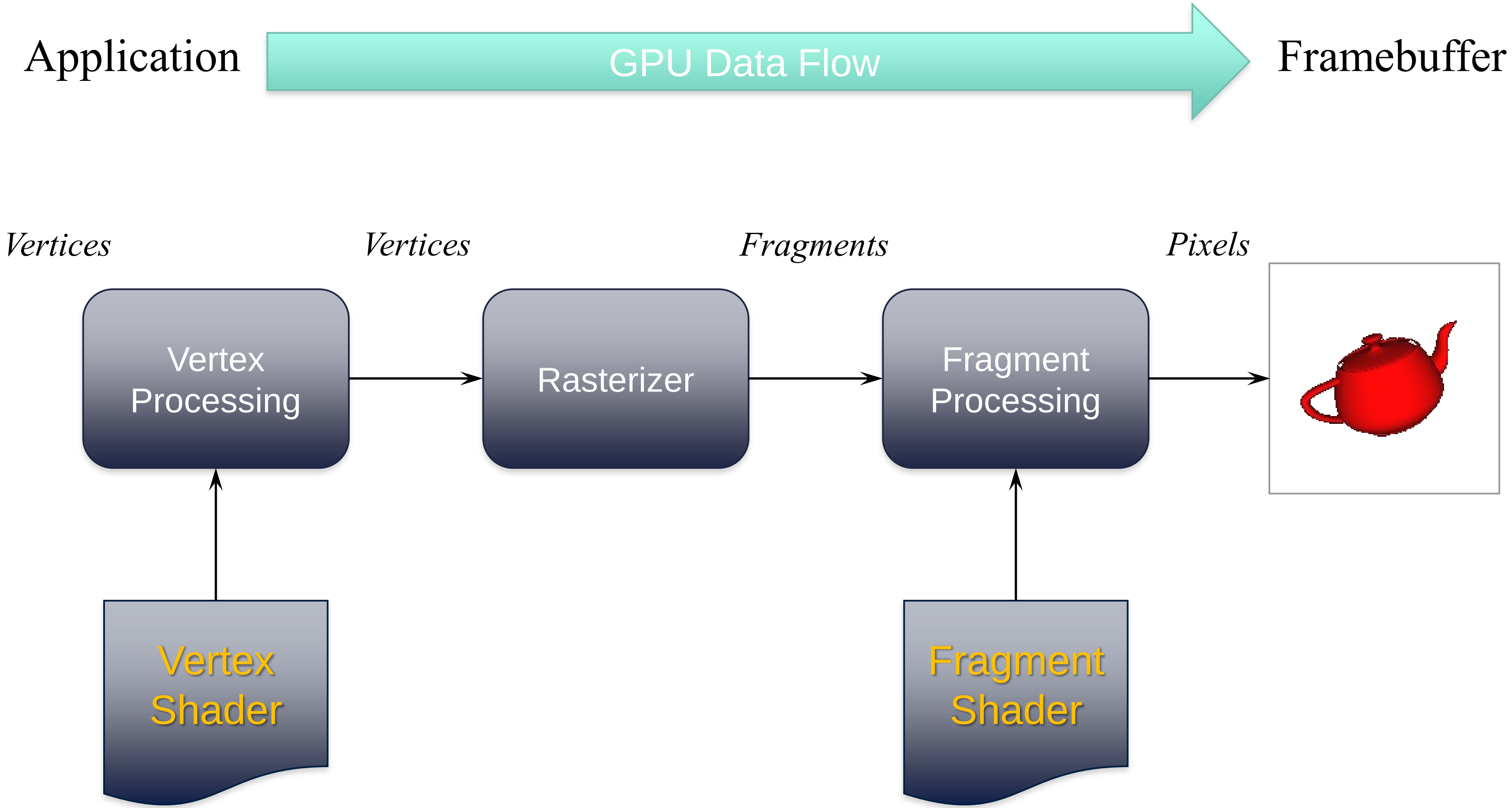
Setting for the most current opengl version for your computer

```
int main(int argc, char ** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
        glutInitWindowSize(512,512);
        glutCreateWindow("Many Points GPU");

        glewExperimental = true;
        glewInit();

        printf("OpenGL %s, GLSL %s\n",
                glGetString(GL_VERSION),
                glGetString(GL_SHADING_LANGUAGE_VERSION));

        glutDisplayFunc(display);
        glutMainLoop();

        return 0;
}
```

For using the modern OpenGL

To check the Current version

OpenGL Pipeline (Simplified)



Application

GPU Data Flow

Framebuffer

Vertices

Vertices

Fragments

Pixels

Vertex Processing

Rasterizer

Fragment Processing

Vertex Shader

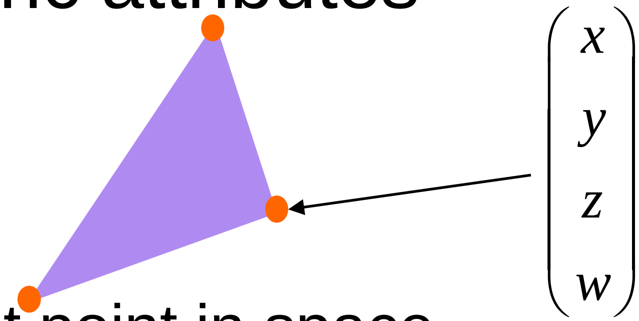
Fragment Shader

OpenGL Programming in a Nutshell

- Modern OpenGL programs essentially do the following steps:
  1. Create buffer objects and load data into them
  2. Create shader programs
  3. “Connect” data locations with shader variables
  4. Render

Representing Geometric Objects

- Geometric objects are represented using vertices
- A vertex is a collection of generic attributes
  - positional coordinates
  - colors
  - texture coordinates
  - any other data associated with that point in space

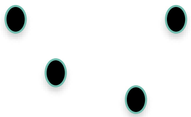


- Position stored in 4 dimensional homogeneous coordinates

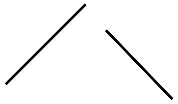
- Vertex data must be stored in *vertex buffer objects* (***VBO***s)
- VBOs must be stored in *vertex array objects* (***VAO***s)

OpenGL’s Geometric Primitives

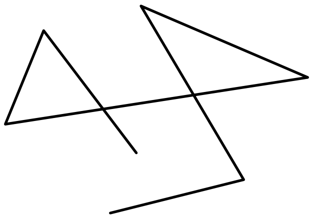
- All primitives are specified by vertices



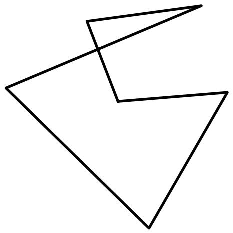
GL_POINTS



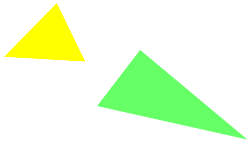
GL_LINES



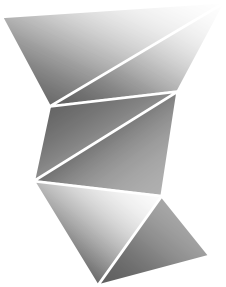
GL_LINE_STRIP



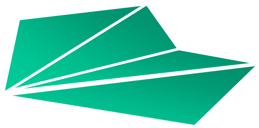
GL_LINE_LOOP



GL_TRIANGLES



GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN

Creating Data

- Define an array for storing all the points

```
struct vec2
{
        float x;
        float y;
};

const int NumPoints = 5000;

void init()
{
        vec2 points[NumPoints];

        for ( int i = 0; i < NumPoints; i++ )
        {
                points[i].x = (rand()%200)/100.0f-1.0f;
                points[i].y = (rand()%200)/100.0f-1.0f;
     }
}
```

Draw the array at once

- Define an array for storing all the points

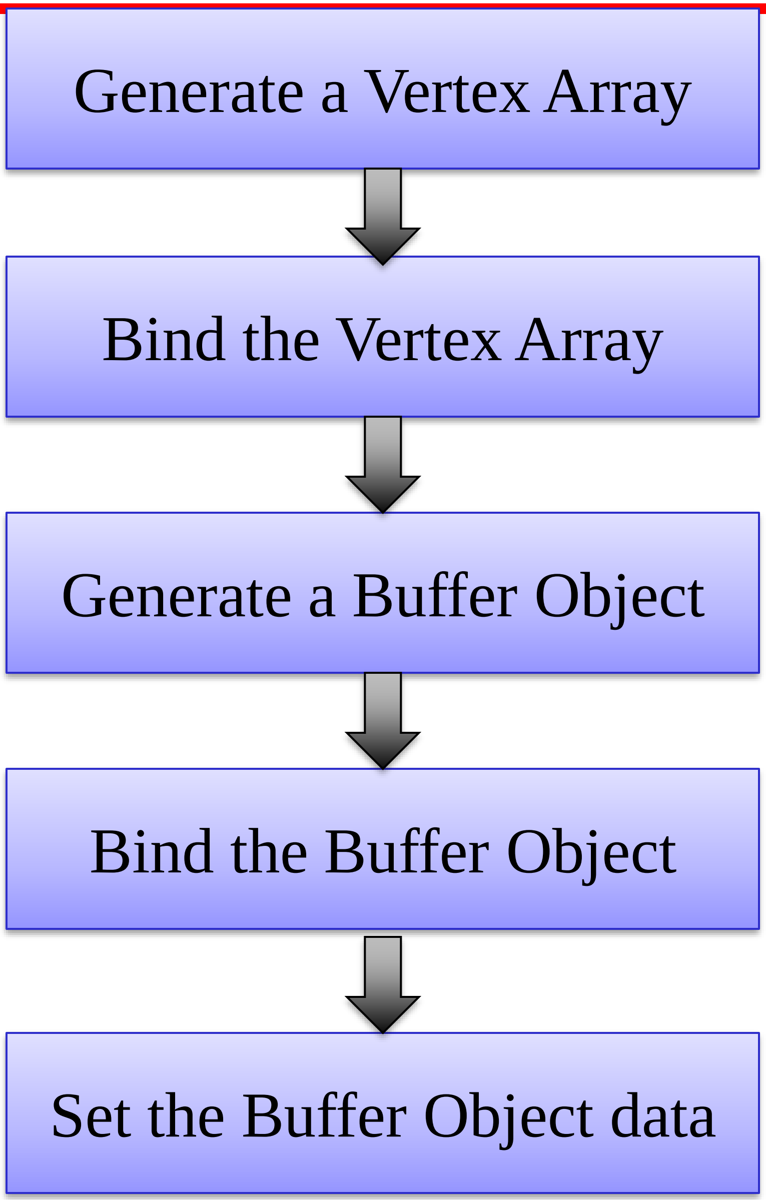
```
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glDrawArrays(GL_POINTS, 0, NumPoints);
}
```

Above code draws the data in GPU.
But we didn’t send the data to GPU at all!!

How to send your data

- Vertex data must be stored in *vertex buffer objects*
  (***VBO***s)
- VBOs must be stored in *vertex array objects*
  (***VAO***s)

How to send your data



Generate a Vertex Array

`glGenVertexArray(..)`

Bind the Vertex Array

`glBindVertexArray(..)`

Generate a Buffer Object

`glGenBuffers(..)`

Bind the Buffer Object

`glBindBuffer(..)`

Set the Buffer Object data

`glBufferData(..)`

Vertex Array Objects (VAOs)

- VAOs store the data of a geometric object
- Steps in using a VAO
  - generate VAO names by calling `glGenVertexArrays()`
  - bind a specific VAO for initialization by calling `glBindVertexArray()`
  - update VBOs associated with this VAO
  - bind VAO for use in rendering
- This approach allows a single function call to specify all the data for an objects
  - previously, you might have needed to make many calls to make all the data current

VAOs in Code

```
// Create a vertex array object
GLuint vao;
glGenVertexArrays(1, &vao);
glBindVertexArray(vao);
```

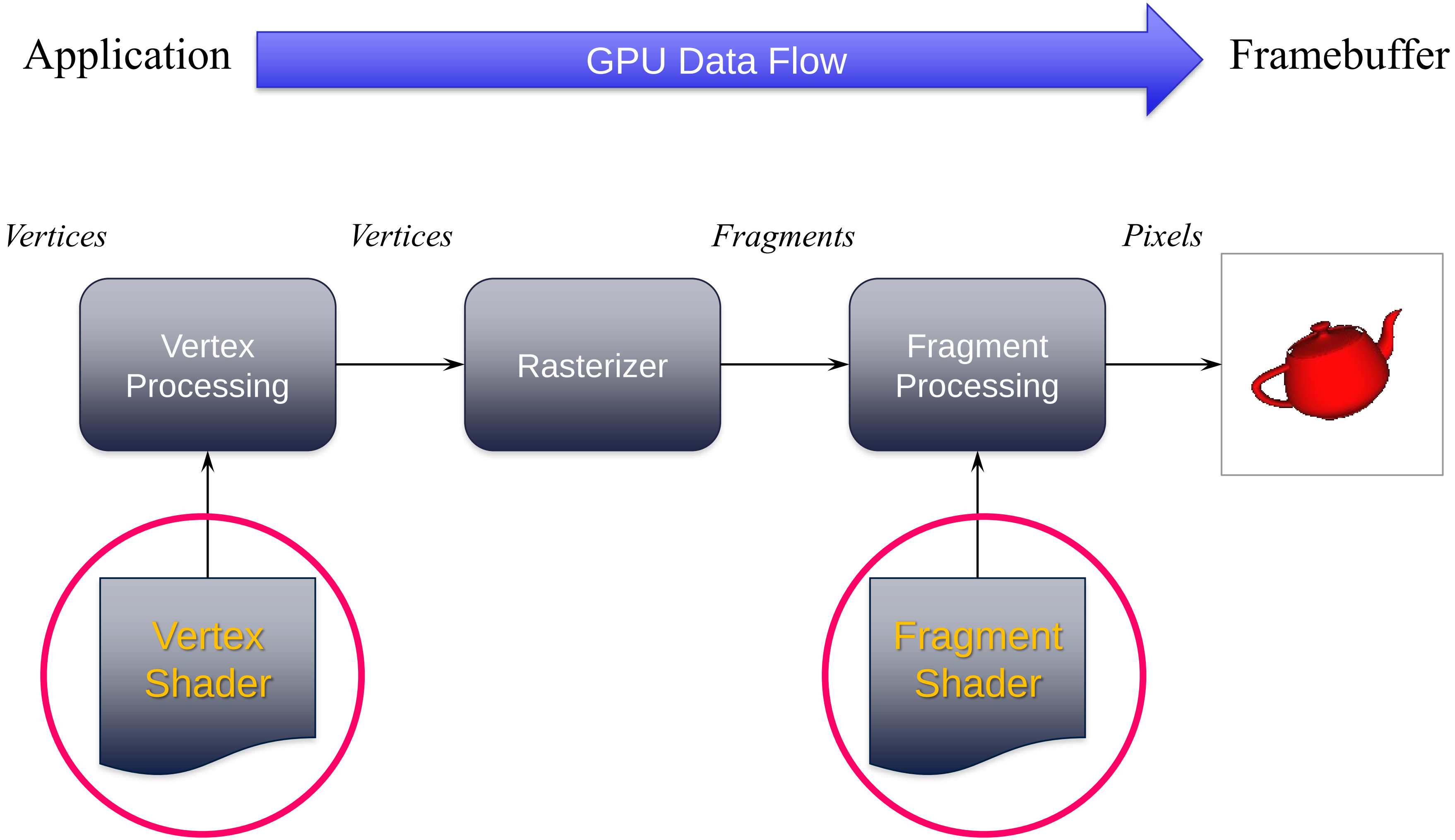
Storing Vertex Attributes

- Vertex data must be stored in a VBO, and associated with a VAO
- The code-flow is similar to configuring a VAO
  - generate VBO names by calling `glGenBuffers()`
  - bind a specific VBO for initialization by calling `glBindBuffer(GL_ARRAY_BUFFER, …)`
  - load data into VBO using `glBufferData(GL_ARRAY_BUFFER, …)`
  - bind VAO for use in rendering later `glBindVertexArray()`

VBOs in Code

```
// Create and initialize a buffer object
GLuint buffer;
glGenBuffers(1, &buffer);
glBindBuffer(GL_ARRAY_BUFFER, buffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(points),
                points, GL_STATIC_DRAW);
```

We need shaders before drawing



Application

GPU Data Flow

Framebuffer

Vertices

Vertices

Fragments

Pixels

Vertex Processing

Rasterizer

Fragment Processing

Vertex Shader

Fragment Shader

Vertex Shader Code (vshader.glsl)

```
#version 330

in vec4 vPosition;

void main()
{
     gl_Position = vPosition;
}
```

Fragment Shader Code (fshader.glsl)

```
#version 330

out vec4 fColor;

void main()
{
        fColor = vec4(1.0,0.0,0.0,1.0);
}
```

Loading Shaders

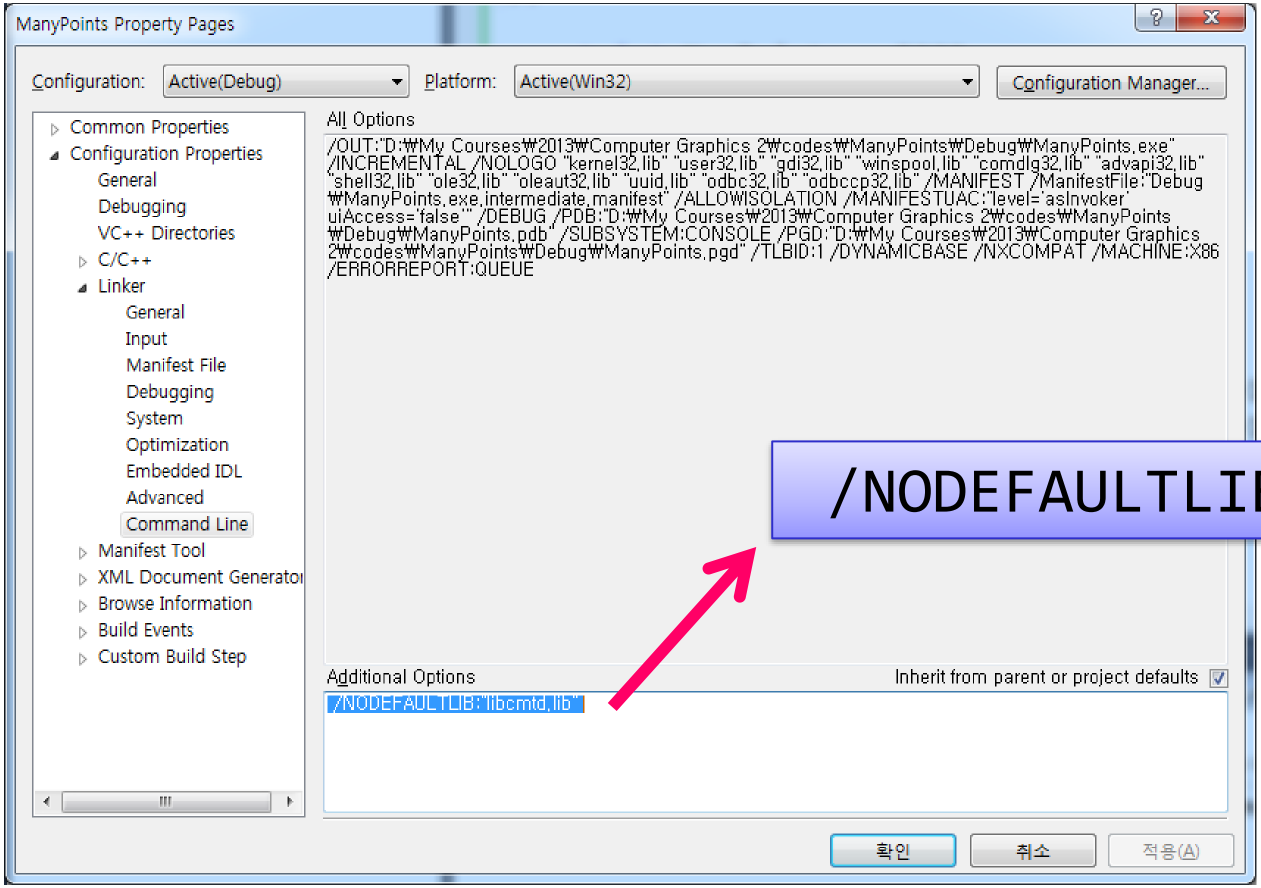
```
#include <InitShader.h>
```

```
// Load and use shaders
GLuint program
     = InitShader( "vshader.glsl", "fshader.glsl" );
glUseProgram( program );
```

glsl : opengl shader language
Those are provided in our homepage.

If you see an error:
You should change Project Setting

- Conflict with an existing lib “libcmtd.lib”:



Configuration: Active(Debug)

Platform: Active(Win32)

All Options

```
/OUT:"D:₩My Courses₩2013₩Computer Graphics 2₩codes₩ManyPoints₩Debug₩ManyPoints.exe"
/INCREMENTAL /NOLOGO "kernel32.lib" "user32.lib" "gdi32.lib" "winspool.lib" "comdlg32.lib" "advapi32.lib"
"shell32.lib" "ole32.lib" "oleaut32.lib" "uuid.lib" "odbc32.lib" "odbccp32.lib" /MANIFEST /ManifestFile:"Debug
₩ManyPoints.exe.intermediate.manifest" /ALLOWISOLATION /MANIFESTUAC:"level='asInvoker'
uiAccess='false'" /DEBUG /PDB:"D:₩My Courses₩2013₩Computer Graphics 2₩codes₩ManyPoints
₩Debug₩ManyPoints.pdb" /SUBSYSTEM:CONSOLE /PGD:"D:₩My Courses₩2013₩Computer Graphics
2₩codes₩ManyPoints₩Debug₩ManyPoints.pgd" /TLBID:1 /DYNAMICBASE /NXCOMPAT /MACHINE:X86
/ERRORREPORT:QUEUE
```

Additional Options

```
/NODEFAULTLIB:"libcmtd.lib"
```

```
/NODEFAULTLIB:"libcmtd.lib"
```

Connecting Vertex Shaders with Geometry

- Application vertex data enters the OpenGL pipeline through the vertex shader
- Need to connect vertex data to shader variables
  - requires knowing the attribute location
- Attribute location can either be queried by calling `glGetVertexAttribLocation()`

Vertex Array Code

```
// set up vertex arrays (after shaders are loaded)
int vPosition = 0;
glEnableVertexAttribArray( vPosition );
glVertexAttribPointer( vPosition, 2, GL_FLOAT,
                GL_FALSE, 0, BUFFER_OFFSET(0) );
```

Drawing Geometric Primitives

- For contiguous groups of vertices

```
glDrawArrays(GL_POINTS, 0, NumPoints);
```

- Usually invoked in display callback
- Initiates vertex shader

Summary

- Setting for libraries
  - Set include/lib folder
  - `#include <vgl.h>`
  - `#include <initshader.h>`
- Creating data (in an array form)
- Sending the data
  - VAO – vertex array object
  - VBO – vertex buffer object
- Loading the shaders
- Draw it with `glDrawArrays(...)`