Debugging

-그래픽스 프로그래밍은 올바르게 렌더링 되지 않았을 경우, 오류의 원인을 파악하기 힘듬

-각 픽셀을 조작하기 때문에 무언가가 작동하지 않을 경우, 오류 로그를 출력할 콘솔이 없고, GLSL코드에 설정할 중단점도 없고, GPU의 실행 상태를 확인할 방법이 없음

-OpenGL이 프로그램을 디버깅 하는 방법에 대해서 알아보기

glGetError()

-OpenGL을 잘못 사용하는 순간(ex) 바인딩하지 않고 버퍼를 구성) 버퍼가 인식되어서 백그라운드에서 하나 이상의 사용자 오류 플래그를 생성, 반환

|  |
| --- |
| GLenum glGetError(); |

-반환하는 오류 플래그

|  |  |  |
| --- | --- | --- |
| 플래그 | 코드 | 설명 |
| GL\_NO\_ERROR | 0 | 오류가 없을 때 |
| GL\_INVALID\_ENUM | 1280 | enum 변수가 적합하지 않을 때 |
| GL\_INVALID\_VALUE | 1281 | value가 적합하지 않을 때 |
| GL\_INVALID\_OPERATION | 1282 | 명령에 대한 상태가 지정된 파라미터에 적합하지 않을 때 |
| GL\_STACK\_OPERFLOW | 1283 | push할 때 오버플로우가 일어났을 떄 |
| GL\_STACK\_UNDERFLOW | 1284 | pop할 때 언더플로우가 일어났을 때 |
| GL\_OUT\_OF\_MEMORY | 1285 | 메모리 할당 작업이 메모리를 할당할 수 없을 때 |
| GL\_INVALID\_FRAMEBUFFER\_OPERATION | 1286 | 완료되지 않은 프레임 버퍼를 읽거나 쓸 |

-OpenGL의 함수 설명서에서 함수가 잘못사용되는 경우의 발생하는 오류 코드를 볼 수 있음

-<https://docs.gl/>

-단, 오류플래그가 설정되는 순간에 다른 오류 플래그는 삭제됨

-또한 glGetError()가 호출되는 순간, 모든 오류 플래그를 지움(분산 시스템의 경우 한개만)

-각 프레임의 끝에서 glGetError을 호출하는 건 이 오류가 유일하다고 생각할 수 없음

|  |
| --- |
| glBindTexture(GL\_TEXTURE\_2D, tex);  std::cout << glGetError() << std::endl; // returns 0 (no error)  glTexImage2D(GL\_TEXTURE\_3D, 0, GL\_RGB, 512, 512, 0, GL\_RGB,  GL\_UNSIGNED\_BYTE, data);  std::cout << glGetError() << std::endl; // returns 1280 (invalid enum)    glGenTextures(-5, textures);  std::cout << glGetError() << std::endl; // returns 1281 (invalid value)    std::cout << glGetError() << std::endl; // returns 0 (no error) |

-glGetError을 사용하면 오류가 발생한 위치를 비교적 쉽게 파악할 수 있음

-화면이 나오지 않을 때, 각 위치에 glGetError을 배치함으로 오류가 처음 나타내는 위치를 빠르게 파악할 수 있음

-glGetError은 반환할 때, 오류 코드를 반환하기 때문에 오류 코드를 알고 있어야함

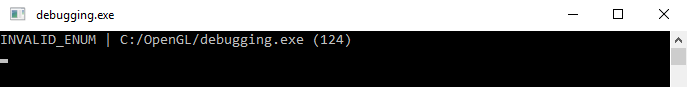
-또는 코드를 이용해서 오류 문자열을 출력할 수 있도록 하면 좋음

|  |
| --- |
| GLenum glCheckError\_(const char \*file, int line)  {  GLenum errorCode;  while ((errorCode = glGetError()) != GL\_NO\_ERROR)  {  std::string error;  switch (errorCode)  {  case GL\_INVALID\_ENUM: error = "INVALID\_ENUM"; break;  case GL\_INVALID\_VALUE: error = "INVALID\_VALUE"; break;  case GL\_INVALID\_OPERATION: error = "INVALID\_OPERATION"; break;  case GL\_STACK\_OVERFLOW: error = "STACK\_OVERFLOW"; break;  case GL\_STACK\_UNDERFLOW: error = "STACK\_UNDERFLOW"; break;  case GL\_OUT\_OF\_MEMORY: error = "OUT\_OF\_MEMORY"; break;  case GL\_INVALID\_FRAMEBUFFER\_OPERATION:  error = "INVALID\_FRAMEBUFFER\_OPERATION"; break;  }  std::cout << error << " | " << file << " (" << line << ")" << std::endl;  }  return errorCode;  }  #define glCheckError() glCheckError\_(\_\_FILE\_\_, \_\_LINE\_\_) |

- \_\_FILE\_\_, \_\_LINE\_\_은 컴파일된 각 파일 및 라인으로 컴파일 시간동안 대체됨, 오류를 반환한 glCheckError 호출을 보다 정확하게 할 수 있음

-예시

|  |
| --- |
| glBindBuffer(GL\_VERTEX\_ARRAY, vbo);  glCheckError(); |

-파일과 line의 위치를 보여주는 것이 확인됨

Debug output in GLFW

-GLFW에서 디버그 context를 요청하는 방법

-glfwCreateWindow를 호출하기 전 작업을 수행하면 됨

|  |
| --- |
| int flags; glGetIntegerv(GL\_CONTEXT\_FLAGS, &flags);  if (flags & GL\_CONTEXT\_FLAG\_DEBUG\_BIT)  {  // initialize debug output  } |

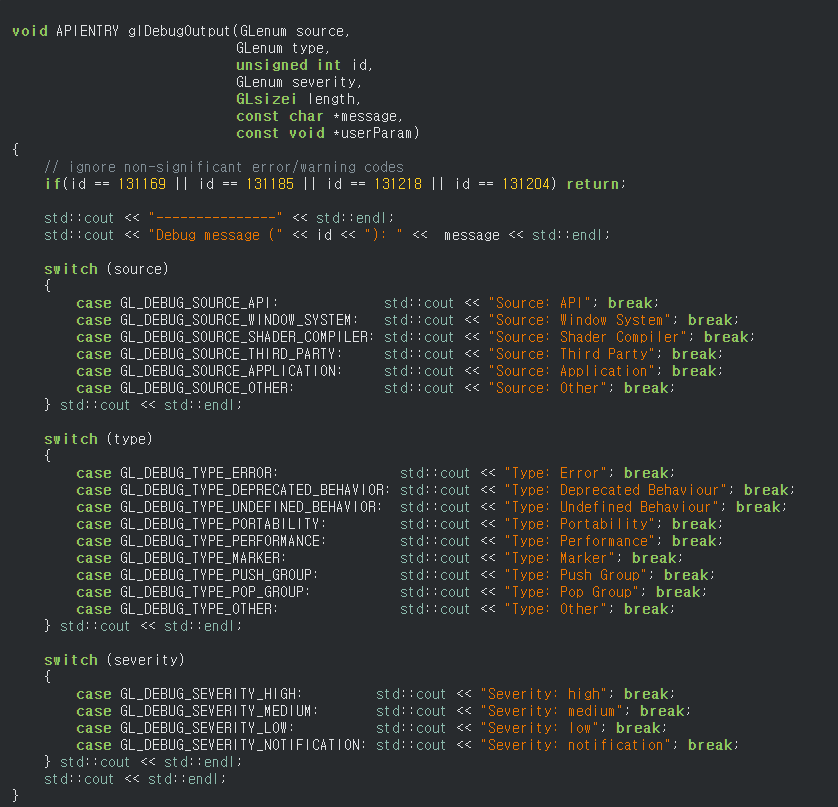
-OpenGL에 오류 로깅 함수 콜백을 전달

-콜백 함수에서는 OpenGL 오류 데이터를 처리

-유용한 오류 데이터를 콘솔에 표시

|  |
| --- |
| void APIENTRY glDebugOutput(GLenum source,  GLenum type,  unsigned int id,  GLenum severity,  GLsizei length,  const char \*message,  const void \*userParam)  {  // ignore non-significant error/warning codes  if(id == 131169 || id == 131185 || id == 131218 || id == 131204) return;  std::cout << "---------------" << std::endl;  std::cout << "Debug message (" << id << "): " << message << std::endl;  switch (source)  {  case GL\_DEBUG\_SOURCE\_API:  std::cout << "Source: API"; break;  case GL\_DEBUG\_SOURCE\_WINDOW\_SYSTEM:  std::cout << "Source: Window System"; break;  case GL\_DEBUG\_SOURCE\_SHADER\_COMPILER:  std::cout << "Source: Shader Compiler"; break;  case GL\_DEBUG\_SOURCE\_THIRD\_PARTY:  std::cout << "Source: Third Party"; break;  case GL\_DEBUG\_SOURCE\_APPLICATION:  std::cout << "Source: Application"; break;  case GL\_DEBUG\_SOURCE\_OTHER:  std::cout << "Source: Other"; break;  } std::cout << std::endl;  switch (type)  {  case GL\_DEBUG\_TYPE\_ERROR:  std::cout << "Type: Error"; break;  case GL\_DEBUG\_TYPE\_DEPRECATED\_BEHAVIOR:  std::cout << "Type: Deprecated Behaviour"; break;  case GL\_DEBUG\_TYPE\_UNDEFINED\_BEHAVIOR:  std::cout << "Type: Undefined Behaviour"; break;  case GL\_DEBUG\_TYPE\_PORTABILITY:  std::cout << "Type: Portability"; break;  case GL\_DEBUG\_TYPE\_PERFORMANCE:  std::cout << "Type: Performance"; break;  case GL\_DEBUG\_TYPE\_MARKER:  std::cout << "Type: Marker"; break;  case GL\_DEBUG\_TYPE\_PUSH\_GROUP:  std::cout << "Type: Push Group"; break;  case GL\_DEBUG\_TYPE\_POP\_GROUP:  std::cout << "Type: Pop Group"; break;  case GL\_DEBUG\_TYPE\_OTHER:  std::cout << "Type: Other"; break;  } std::cout << std::endl;    switch (severity)  {  case GL\_DEBUG\_SEVERITY\_HIGH:  std::cout << "Severity: high"; break;  case GL\_DEBUG\_SEVERITY\_MEDIUM:  std::cout << "Severity: medium"; break;  case GL\_DEBUG\_SEVERITY\_LOW:  std::cout << "Severity: low"; break;  case GL\_DEBUG\_SEVERITY\_NOTIFICATION:  std::cout << "Severity: notification"; break;  } std::cout << std::endl;  std::cout << std::endl;  } |

-가독성용 사진

-OpenGL에서 오류가 감지될 때 마다 콜백 함수를 호출해서 OpenGL 오류와 관련된 많은 정보를 출력할 수 있음

-여기서 NVidia 드라이버의 131185와 같은 버퍼가 생성되었음을 알려주는 유용한 내용을 표시하지 않는 몇 가지 오류 코드는 무시하였음

-콜백 함수를 사용할 수 있으니 디버그 출력을 초기화

|  |
| --- |
| if (flags & GL\_CONTEXT\_FLAG\_DEBUG\_BIT)  {  //디버그 출력을 사용하도록 지시  glEnable(GL\_DEBUG\_OUTPUT);  //오류가 발생한 순간 콜백 함수를 호출  glEnable(GL\_DEBUG\_OUTPUT\_SYNCHRONOUS);  glDebugMessageCallback(glDebugOutput, nullptr);  glDebugMessageControl(GL\_DONT\_CARE, GL\_DONT\_CARE, GL\_DONT\_CARE,  0, nullptr, GL\_TRUE);  } |

Filter debug output

-glDebugMessageControl을 사용하면 메시지를 수신하려는 오류 유형을 필터링할 수 있음

-OpenGL API 의 메시지의 오류와 심각도만 알리기 위해 다음과 같이 구성함

|  |
| --- |
| glDebugMessageControl(GL\_DEBUG\_SOURCE\_API,  GL\_DEBUG\_TYPE\_ERROR,  GL\_DEBUG\_SEVERITY\_HIGH,  0, nullptr, GL\_TRUE); |

