Advanced OpenGL // Advanced Data

-텍스처를 동해 많은 양의 데이터를 shader에 전달하는 버퍼, 또는 메소드를 조작하는데 더 흥미로운 방법이 있음

-보다 새로운 버퍼 함수와 텍스처 객체를 사용해서 많은 양의 데이터를 저장하는 방법에 대해 배워볼 것

-OpenGL 버퍼는 GPU 메모리의 특정 부분만 관리하는 객체, 특정 버퍼에 바인딩할 때, 의미를 부여함

-버퍼는 GL\_ARRAY\_BUFFER에 바인딩 하면 정점 배열 버퍼일 뿐인데, 이것은 우리가 GL\_ELEMENT\_ARRAY\_BUFFER에 게 바인딩 할 수 있음

-OpenGL은 내부적으로 타겟당 버퍼를 저장하고, 타겟을 기반으로 다르게 처리함

-지금까지 메모리 객체를 할당, 메모리에 데이터를 추가하는 glBufferData를 호출해서 버퍼 객체가 관리하는 메모리를 채웠음

-이때, 채우는 데이터를 NULL로 전달하면 메모리만 할당하고 데이터는 채우지 않음

-이 버퍼는 특정 양의 메모리를 예약하고 이 버퍼로 돌아와 fragment별로 채우기를 원할 때 유용하게 사용되어짐

glBufferSubData

-버퍼의 특정 영역을 채울 수 있는 함수

-버퍼 타겟, 오프셋, 데이터 크기, 실제 데이터를 인수로 받음

-버퍼를 채우려는 위치를 지정하는 오프셋을 지정할 수 있다는게 다른 점

-이를 이용해서 버퍼 메모리의 특정 부분만을 삽입/수정 할 수 있음

-호출하기 전, glBufferData에 대한 호출이 필요하도록 버퍼에 충분한 메모리가 있어야함

|  |
| --- |
| glBufferSubData(GL\_ARRAY\_BUFFER, 24, sizeof(data), &data);  // 범위 : [24, 24 + sizeof(data)] |

-버퍼에 데이터를 가져오는 다른 방법은 버퍼의 메모리에 대한 포인터를 요청, 버퍼에 데이터를 복사하는 것

-glMapBuffer을 이용해서 바인드된 버퍼의 메모리에 대한 포인터를 반환

|  |
| --- |
| float data[] = {  0.5f, 1.0f, -0.35f  [...]  };  glBindBuffer(GL\_ARRAY\_BUFFER, buffer);  // 포인터 가져오기  void \*ptr = glMapBuffer(GL\_ARRAY\_BUFFER, GL\_WRITE\_ONLY);  // 메모리에 데이터 복사  memcpy(ptr, data, sizeof(data));  // 포인터 사용의 종료를 알림  glUnmapBuffer(GL\_ARRAY\_BUFFER); |

-매핑을 해제하면 포인터가 유효하지 않게 됨

-버퍼에 데이터를 성공적으로 매핑할 수 있으면 GL\_TRUE를 반환함

-MapBuffer을 사용하면 임시 메모리를 사용하지 않고 직접 버퍼에 매핑함

Batching vertex attributes

-지금까지 glVertexAttribPointer을 사용해서 각 정점의 위치, 법선, 텍스처 좌표를 옆에 나열했었음

-버퍼에 대해서 공부했으니 새로운 접근법을 사용해 보자

-지금까지 사용한 방법은 123123123123 이였는데, 이를 1111 2222 3333 으로 일괄처리(Batch)해서 사용

|  |
| --- |
| float positions[] = { ... };  float normals[] = { ... };  float tex[] = { ... };  // fill buffer  glBufferSubData(GL\_ARRAY\_BUFFER, 0, sizeof(positions), &positions);  glBufferSubData(GL\_ARRAY\_BUFFER, sizeof(positions), sizeof(normals), &normals);  glBufferSubData(GL\_ARRAY\_BUFFER, sizeof(positions) + sizeof(normals), sizeof(tex), &tex); |

-하나의 커다란 배열로 전송해도 가능하나 이런식으로 배열을 나눠서 glBufferSubData를 사용하면 완벽하게 처리할 수 있고, 가시성, 수정면에서 더 좋을 것 같이 보임

-또한 glVertexAttribPointer을 업데이트 해야함

|  |
| --- |
| glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 3 \* sizeof(float), 0);  glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE,  3 \* sizeof(float), (void\*)(sizeof(positions)));  glVertexAttribPointer(2, 2, GL\_FLOAT, GL\_FALSE,  2 \* sizeof(float), (void\*)(sizeof(positions) + sizeof(normals))); |

-이 방법은 OpenGL에 즉각적인 이점은 없음

-그러나 vertex 속성을 정의하는데 보다 체계적인 방법임

Copying Buffers

-데이터로 채워진 버퍼는 다른 버퍼와 공유하거나 복사할 수 있음

-glCopyBufferData

|  |
| --- |
| void glCopyBufferSubData(GLenum readtarget, GLenum writetarget,  GLintptr readoffset, GLintptr writeoffset, GLsizeiptr size); |

-예를 들어 VAO에서 EBO로 각각 읽기 및 쓰기 타겟으로 지정해 복사할 수 있음

-그러면 해당 버퍼 대상에 바인드된 버퍼가 영향을 받음

-두개의 정점 배열 버퍼에 데이터를 읽고 쓰고 싶다면 어떻게 해야 하는가

-같은 버퍼 타겟에 동시에 2개의 버퍼를 바인드 할 수 없음

-GL\_COPY\_READ\_BUFFER, GL\_COPY\_WRITE\_BUFFER라는 두개의 버퍼 대상을 추가로 제공해줌

-선택한 버퍼를 새로운 버퍼 타겟에 바인드하고, 해당 타겟을 readtarget이나 writetarget 인수로 설정하면 됨

-이렇게 사용하면 readtarget의 readoffset에서 주어진 크기의 데이터를 읽어서 writetarget의 버퍼에 writeoffset에서 쓰기를 함

|  |
| --- |
| glBindBuffer(GL\_COPY\_READ\_BUFFER, vbo1);  glBindBuffer(GL\_COPY\_WRITE\_BUFFER, vbo2);  glCopyBufferSubData(GL\_COPY\_READ\_BUFFER, GL\_COPY\_WRITE\_BUFFER,  0, 0, 8 \* sizeof(float)); |
| float vertexData[] = { ... };  glBindBuffer(GL\_ARRAY\_BUFFER, vbo1);  glBindBuffer(GL\_COPY\_WRITE\_BUFFER, vbo2);  glCopyBufferSubData(GL\_ARRAY\_BUFFER, GL\_COPY\_WRITE\_BUFFER,  0, 0, 8 \* sizeof(float)); |