# 정점 버퍼 객체와 인덱스 버퍼 객체 활용하기

김준호

**Abstract**

모던 OpenGL의 서버-클라이언트 모델을 이해하고 모던 OpenGL 프로그래밍의 원리를 파악한다. 버퍼 객체를 이용해 물체를 그리는 간단한 모던 OpenGL 프로그래밍을 작성함으로써 OpenGL 버퍼 객체의 활용법을 이해한다. 정점의 삼차원 위치, 색깔 등과 같은 정점 속성(vertex attribute)을 GPU 메모리에 저장하는 정점 버퍼 객체 (vertex buffer object, VBO)를 설정하고 활용하는 방법을 학습한다. 폴리곤을 이루는 정점의 인덱스를 GPU 메모리에 저장하는 인덱스 버퍼 객체 (index buffer object, IBO)를 설정하고 활용하는 방법을 학습한다. GPU 메모리에 저장된 데이터를 이용해 모던 OpenGL로 화면에 물체를 그리는 방법을 이해한다.

## 1 모던 OpenGL 프로그래밍 원리

모던 OpenGL은 클라이언트-서버 프로그래밍 모델에 기반을 두고 있다. 여기서는 모던 OpenGL의 클라이언트-서버 모델을 이해하고, 어떤 방식으로 모던 OpenGL에서 물체를 그리기 위한 정점 데이터가 관리되어야 하는지 학습한다.

- 정점에 대한 위치(position), 색깔(color)과 같은 정점 속성(vertex attribute) 데이터를 클라이언트 메모리 측에서 설정하고 서버 메모리 측에 전송하는 방법을 학습한다.
- 서버 메모리에 정점 버퍼 객체(vertex buffer object, 이하 VBO)를 정의하고, 클라이언트 메모리에 설정된 정점 속성을 서버 메모리에 전송하는 방법을 학습한다.
- 서버 메모리에 정점 인덱스 객체(index buffer object, 이하 IBO)를 정의하고, 클라이언트 메모리에 설정된 폴리곤의 정점 인덱스를 서버 메모리에 전송하는 방법을 학습한다.
- 서버 메모리에 정의된 VBO와 IBO를 이용해 모던 OpenGL로 화면을 그리는 방법을 학습한다.

### 1.1 클라이언트-서버 모델

모던 OpenGL에서는 프로그래밍 가능한 렌더링 파이프라인을 지원하기 위해 Fig. 1과 같은 클라이언트-서버 모델을 따르는 프로그래밍 기법을 도입하고 있다.

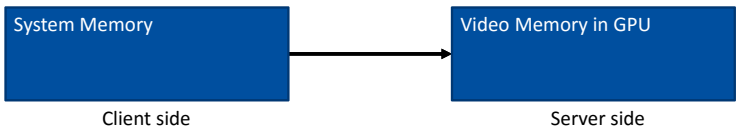


Fig. 1. OpenGL 클라이언트-서버 모델

클라이언트는 CPU와 메인메모리로 이루어지는 일반적인 컴퓨팅 환경을 의미한다. 일반적으로 클라이언트 측에서 돌아가는 코드는 C/C++로 작성된 OpenGL 코드이며, 때에 따라 Java/JavaScript 등 다른 언어로 작성될 수도 있다.

서버는 GPU와 GPU 메모리로 이루어지는 그래픽카드 상의 컴퓨팅 환경을 의미한다. 서버 측에서 돌아가는 코드는 OpenGL 쉐이딩 언어(OpenGL Shading Language, 이하 GLSL)로 작성된 쉐이더 코드이다. 지금은 우선 쉐이더 코드에 대해 다루지 않고, OpenGL의 클라이언트-서버 모델에 대해서만 집중하도록 하자. 쉐이더 코드를 작성하고 그 내용을 이해하는 부분은 학기 후반에 다룰 예정이다.

모던 OpenGL이 도입하고 있는 클라이언트-서버 모델은 다음과 같은 방식으로 동작한다.

- CPU는 메인 메모리에 적힌 내용만 볼 수 있다.
- GPU는 GPU 메모리에 적힌 내용만 볼 수 있다.
- 메인 메모리에 적힌 내용을 GPU 메모리에 전송하려면 특정 모던 OpenGL 명령들을 사용해야 한다.
- GPU는 GPU 메모리에 적힌 내용으로 이용해 화면을 렌더링 한다.

CPU와 GPU는 각자 자신이 볼 수 있는 메모리만 본다는 것이 중요한 점이다. 따라서, 모던 OpenGL을 이용해 삼각형을 그리려면 삼각형의 정점 데이터가 메인 메모리와 GPU 메모리 두 곳에 모두 존재해야 한다.

*국민대학교 소프트웨어학부, junho@kookmin.ac.kr*

## 2 정점 버퍼 객체(VBO) 이해하기

물체를 triangle soup 방식으로 그릴 때, GPU 메모리를 활용하는 방식인 정점 버퍼 객체(vertex buffer object, VBO)를 살펴보도록 한다.

VBO를 이용해 삼각형을 그리기 위한 핵심코드는 아래 세 부분이다.

1) 메인 메모리에 정점 데이터를 정의하는 부분
2) GPU 메모리에서 VBO를 만들고, 메인 메모리에 정의된 정점 데이터를 GPU 메모리의 VBO로 전송하는 부분
3) GPU 메모리의 VBO에 정의된 정점 데이터로 물체를 그리는 부분

### 2.1 정점 데이터 정의 (CPU 메모리)

정점(vertex) 삼차원 위치(position)와 색깔(color)을 정의하는 코드 조각이다. 여기서는 삼각형 2개를 이용해 사각형 1개를 그리고자 한다. 다음 코드 조각과 같이 6개의 정점에 대한 위치와 칼라 데이터를 표현하도록, 배열 `g_position[]`과 배열 `g_color[]`를 구성하였다.

```
// per-vertex 3D positions (x, y, z)
GLfloat g_position[] = {
   0.5f,  0.5f,  0.0f,     // 0th vertex position
  -0.5f, -0.5f,  0.0f,     // 1st vertex position
   0.5f, -0.5f,  0.0f,     // 2nd vertex position

   0.5f,  0.5f,  0.0f,     // 3nd vertex position
  -0.5f,  0.5f,  0.0f,     // 4nd vertex position
  -0.5f, -0.5f,  0.0f,     // 5nd vertex position
};
// per-vertex RGB color (r, g, b)
GLfloat g_color[] = {
  1.0f, 0.0f, 0.0f,        // 0th vertex color (red)
  0.0f, 1.0f, 0.0f,        // 1st vertex color (green)
  0.0f, 0.0f, 1.0f,        // 2nd vertex color (blue)

  1.0f, 0.0f, 0.0f,        // 3nd vertex color (red)
  1.0f, 1.0f, 0.0f,        // 4nd vertex color (yellow)
  0.0f, 1.0f, 0.0f,        // 5nd vertex color (green)
};
```

### 2.2 정점 데이터 전송 (CPU 메모리 → GPU 메모리)

CPU 메모리에 있는 정점 데이터를 어떻게 GPU 메모리로 전송하는지 살펴보자. 먼저 정점 데이터 전송에 앞서 GPU 메모리에 정점 데이터가 보관된 장소를 마련해야 한다. 모던 OpenGL에서는 정점 데이터가 보관될 GPU 메모리를 할당하는 특수한 방법을 제공하는데 이를 정점 버퍼 객체(vertex buffer object, 이하 VBO)라고 한다.

다음은 GPU 메모리에 정점의 삼차원 위치와 칼라를 담을 VBO를 2개 설정하고, CPU 메모리의 정점 정보를 GPU 메모리 상의 VBO로 전송하는 코드 조각이다.

```
GLuint  position_buffer;
GLuint  color_buffer;

void init_buffer_objects()
{
  glGenBuffers(1, &position_buffer);
  glBindBuffer(GL_ARRAY_BUFFER, position_buffer);
  glBufferData(GL_ARRAY_BUFFER, sizeof(g_position), g_position, GL_STATIC_DRAW);

  glGenBuffers(1, &color_buffer);
  glBindBuffer(GL_ARRAY_BUFFER, color_buffer);
  glBufferData(GL_ARRAY_BUFFER, sizeof(g_color), g_color, GL_STATIC_DRAW);
}
```

위의 코드 조각에서 눈여겨 봐야할 모던 OpenGL 함수는 다음과 같다.

- `glGenBuffers`: GPU 메모리에 버퍼 객체를 만들어 달라고 요구하는 함수이다. 만들어진 버퍼 객체에 접근할 수 있는 인덱스는 call-by-reference 방식으로 넘겨 준다. 한번에 여러개를 만들어 달라고 요청할 수 있지만, 이 코드 조각에서는 `glGenBuffers`이 호출될 때마다 1개씩 만들어 달라고 요청하였다.

- `glBindBuffer`: GPU 메모리에 생성된 버퍼 객체를 바인드하는 함수이다. `glBindBuffer`의 함수인자로 `GL_ARRAY_BUFFER`를 지정함으로써, GPU 메모리의 버퍼 객체를 정점 속성(vertex attributes)을 저장할 수 있는 배열로 활용하였다.
- `glBufferData`: CPU 메모리에 있는 데이터를 GPU 메모리로 전송시키는 함수이다. 현재 바인드 되어 있는 버퍼 객체로 데이터가 전송된다.

### 2.3 정점 데이터로 삼각형 그리기 (GPU 메모리)

정점 데이터가 모두 GPU 메모리에 전송된 상태이므로 모던 OpenGL은 GPU 메모리에 있는 VBO에 기록된 삼각형 정점 데이터를 참조하여 그림을 그릴 수 있다.

다음은 GPU의 VBO에 저장된 정점 속성을 이용해 triangle soup 방식으로 물체를 그리는 코드이다.

```
void render_object()
{
  // glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  // 특정 쉐이더 프로그램 사용
  glUseProgram(program);

  // 정점 attribute 배열 loc_a_position 활성화
  glEnableVertexAttribArray(loc_a_position);
  // 앞으로 언급하는 배열 버퍼(GL_ARRAY_BUFFER)를 position_buffer로 지정
  glBindBuffer(GL_ARRAY_BUFFER, position_buffer);
  // 현재 배열 버퍼에 있는 데이터를 버텍스 쉐이더 loc_a_position 위치의 attribute와 연결
  glVertexAttribPointer(loc_a_position, 6, GL_FLOAT, GL_FALSE, 0, (void*)0);

  // 정점 attribute 배열 loc_a_color 활성화
  glEnableVertexAttribArray(loc_a_color);
  // 앞으로 언급하는 배열 버퍼(GL_ARRAY_BUFFER)를 color_buffer로 지정
  glBindBuffer(GL_ARRAY_BUFFER, color_buffer);
  // 현재 배열 버퍼에 있는 데이터를 버텍스 쉐이더 loc_a_color 위치의 attribute와 연결
  glVertexAttribPointer(loc_a_color, 6, GL_FLOAT, GL_FALSE, 0, (void*)0);

  // 삼각형 그리기
  glDrawArrays(GL_TRIANGLES, 0, 6);

  // 정점 attribute 배열 비활성화
  glDisableVertexAttribArray(loc_a_position);
  glDisableVertexAttribArray(loc_a_color);
  // 쉐이더 프로그램 사용해제
  glUseProgram(0);
}
```

이 코드 조각에서는 정점의 삼차원 위치가 담긴 VBO와 칼라가 담긴 VBO를 각각 쉐이더(shader)의 특정 속성(attribute)에 결합시키고 삼각형을 렌더링하고 있다.

여기서 눈여겨 봐야할 부분은 `glVertexAttribPointer`의 가장 마지막 함수인자로 `(void*)0`를 넘겨줬다는 점이다. 원래 이 부분은 정점 데이터가 있는 CPU 메모리의 주소를 알려주는 부분인데, 우리는 GPU 메모리에 복사된 정점 데이터를 활용하기로 했으므로 CPU 메모리는 활용하지 않겠다는 의미에서 `(void*)0`로 설정했다는 점을 주목하자. 그렇다면, 정점의 속성 정보가 있는 GPU 메모리의 위치는 어떻게 알려준 걸까? 가장 최근에 `glBindBuffer`로 바인드된 VBO가 `glVertexAttribPointer`에서 활용하는 정점 속성을 담고 있는 GPU 메모리 위치로 활용된다.

정리하자면, `glVertexAttribPointer`에서 가장 마지막 함수인자가 `(void*)0`인 경우, 정점 속성 데이터를 얻기 위해 CPU 메모리를 참조하지 않고 가장 최근에 바인드된 VBO를 참조하는 방식으로 동작한다.

## 3 인덱스 버퍼 객체(IBO) 이해하기

이제 물체를 vertex list & triangles 방식으로 그릴 때, GPU 메모리를 활용하는 방식인 정점 버퍼 객체(vertex buffer object, VBO)와 인덱스 버퍼 객체(index buffer object, IBO)를 살펴보도록 한다.

VBO의 경우, 정점 속성을 메모리에 저정할 때 vertex list & triangles 방식으로 관리해야한다는 점을 제외하고는 앞서 설명한 방식과 동일하다. 여기서는 CPU 메모리에 정의된 IBO를 어떻게 GPU 메모리에 올리고 활용하는지에 대한 핵심코드만 살펴보기로 한다. IBO를 이용해 폴리곤 정점의 인덱스를 표현하는 핵심코드는 아래 세부분이다.

1) 메인 메모리에 폴리곤 정점 인덱스 데이터를 정의하는 부분
2) GPU 메모리에서 IBO를 만들고, 메인 메모리에 정의된 폴리곤 정점 인덱스 데이터를 GPU 메모리의 IBO로 전송하는 부분
3) GPU 메모리의 IBO에 정의된 폴리곤 정점 인덱스를 활용해 물체를 그리는 부분

### 3.1 인덱스 데이터 정의 (CPU 메모리)

삼각형을 이루는 정점 인덱스를 정의하는 코드 조각이다. 삼각형 2개를 그리기 위해, 총 6개의 인덱스 정보를 CPU 메모리 상의 배열에 나타내었다. 정점 데이터의 경우, vertex list & triangle 방식으로 그리기 위해 4개의 위치와 칼라 정보를 정의하도록 수정하였다.

```
// triangle-vertex indices
GLubyte g_indices[] = {
   0, 1, 2, 0, 3, 1,
};

// vertex list & triangles 방식으로 그리기 위해 정점 데이터를 수정함.
// per-vertex 3D positions (x, y, z)
GLfloat g_position[] = {
   0.5f,  0.5f,  0.0f,    // 0th vertex position
  -0.5f, -0.5f,  0.0f,    // 1st vertex position
   0.5f, -0.5f,  0.0f,    // 2nd vertex position
  -0.5f,  0.5f,  0.0f,    // 3rd vertex position
};
// per-vertex RGB color (r, g, b)
GLfloat g_color[] = {
  1.0f, 0.0f, 0.0f,       // 0th vertex color (red)
  0.0f, 1.0f, 0.0f,       // 1st vertex color (green)
  0.0f, 0.0f, 1.0f,       // 2nd vertex color (blue)
  1.0f, 1.0f, 0.0f,       // 3rd vertex color (yellow)
};
```

### 3.2 인덱스 데이터 전송 (CPU 메모리 → GPU 메모리)

CPU 메모리에 있는 정점 데이터를 어떻게 GPU 메모리로 전송하는지 살펴보자. 먼저 정점 데이터 전송에 앞서 GPU 메모리에 정점 데이터가 보관된 장소를 마련해야 한다. 모던 OpenGL에서는 정점 데이터가 보관될 GPU 메모리를 할당하는 특수한 방법을 제공하는데 이를 정점 버퍼 객체(vertex buffer object, 이하 VBO)라고 한다.

다음은 GPU 메모리에 정점의 삼차원 위치와 칼라를 담을 VBO를 2개 설정하고, CPU 메모리의 정점 정보를 GPU 메모리 상의 VBO로 전송하는 코드 조각이다.

```
GLuint  index_buffer;

void init_buffer_objects()
{
  // VBO 정의하는 부분 (2.2절의 예제와 동일, 생략)

  // IBO 정의하는 부분
  glGenBuffers(1, &index_buffer);
  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, index_buffer);
  glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(g_indices), g_indices, GL_STATIC_DRAW);
}
```

위의 코드 조각에서 `glBindBuffer`와 `glBufferData`의 함수인자로 `GL_ELEMENT_ARRAY_BUFFER`를 지정했다는 점에 주목하도록 하자. 이렇게 지정함으로써 버퍼 객체가 인덱스 데이터를 담아두는 장소임을 명시할 수 있다. 나머지 부분은 이전 예제와 유사하다.

### 3.3 VBO와 IBO를 이용해 삼각형 그리기 (GPU 메모리)

정점 데이터와 인덱스 데이터가 각각 VBO와 IBO 형태로 GPU 메모리에 전송된 상태이므로, 모던 OpenGL에서는 다음의 코드 조각과 같이 VBO와 IBO를 이용해 vertex list & triangles 방식으로 물체를 그릴 수 있다.

```
void render_object()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  // 특정 쉐이더 프로그램 사용
  glUseProgram(program);
```

```
  // VBO 설정
  glEnableVertexAttribArray(loc_a_position);
  glBindBuffer(GL_ARRAY_BUFFER, position_buffer);
  glVertexAttribPointer(loc_a_position, 4, GL_FLOAT, GL_FALSE, 0, (void*)0);

  glEnableVertexAttribArray(loc_a_color);
  glBindBuffer(GL_ARRAY_BUFFER, color_buffer);
  glVertexAttribPointer(loc_a_color, 4, GL_FLOAT, GL_FALSE, 0, (void*)0);

  // IBO를 이용해 물체 그리기
  // glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, g_indices); // CPU 메모리의 인덱스 데이터 활용
  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, index_buffer);
  glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_BYTE, (void*)0); // GPU 메모리의 IBO 활용

  // 정점 attribute 배열 비활성화
  glDisableVertexAttribArray(loc_a_position);
  glDisableVertexAttribArray(loc_a_color);
  // 쉐이더 프로그램 사용해제
  glUseProgram(0);
}
```

이 코드 조각에서는 정점의 삼차원 위치가 담긴 VBO와 칼라가 담긴 VBO를 각각 쉐이더(shader)의 특정 속성(attribute)에 결합시키고 삼각형을 렌더링하고 있다.

먼저, vertex list & triangles 방식을 쓰기 때문에 VBO에서 저장하는 정점의 속성 개수가 달라졌다는 것도 유의할 사항이다. `glVertexAttribPointer`의 함수인자로 정점의 속성 개수가 `4`로 변했음에 주목하자.

그 다음 눈여겨 봐야할 부분은 `glDrawElements`가장 마지막 함수인자로 `(void*)0`을 넘겨줬다는 점이다. 원래 이 부분은 인덱스 데이터가 있는 CPU 메모리의 주소를 알려주는 부분인데, 우리는 GPU 메모리 상의 IBO 데이터를 활용하기로 했으므로 CPU 메모리는 활용하지 않겠다는 의미에서 `(void*)0`로 설정했다는 점을 주목하자. 그렇다면, 인덱스 정보가 있는 GPU 메모리의 위치는 어떻게 알려준 걸까? 가장 최근에 `glBindBuffer`로 바인드된 IBO가 `glDrawElements`에서 활용하는 인덱스 정보를 담고 있는 GPU 메모리 위치로 활용된다.

정리하자면, `glDrawElements`에서 가장 마지막 함수인자가 `(void*)0`인 경우, 인덱스 데이터를 얻기 위해 CPU 메모리를 참조하지 않고 가장 최근에 바인드된 IBO를 참조하는 방식으로 동작한다.

## 4 정리

모던 OpenGL에서 버퍼 객체를 이용해 렌더링을 할 때, 메인 메모리의 데이터를 전혀 참조하지 않고 GPU 메모리에 있는 데이터만 참조해서 그림을 그렸다는 사실이다. 모던 OpenGL이 이런 방식으로 동작하기 때문에 정점에 대한 데이터가 메인 메모리와 GPU 메모리 두 곳에 존재한다는 사실 또한 중요하다. 따라서, 효과적인 모던 OpenGL 프로그램 작성을 위해서는 두 곳에 존재하는 데이터를 어떻게 효과적으로 동기화(synchronization)할 것인가를 항상 고민해야 한다.

## 5 생각해 볼 문제

모던 OpenGL의 클라이언트-서버 모델을 이해하고 다음 사항을 고민해 보자.

1) 정점의 삼차원 위치나 칼라 데이터를 업데이트하고 새로 그리려면 어떤 방식으로 해야 할까?
2) GPU가 메인 메모리를 접근하지 못하도록 설계되어 있는데 왜 그런 것일까?

참고: `glMapBuffer`와 `glUnmapBuffer`를 활용하면 위 사항에 대한 고민을 해결할 수 있다.