사용자 입력에 반응하는 그래픽스 프로그래밍

김준호

Abstract

사용자 입력에 반응하는 그래픽스 프로그램을 작성하는 기법을 익힌다. 사용자 입력을 통해 물체의 위치를 바꾸는 방법을 이해한다. GLFW의 콜백함수(callback function)을 이용해 윈도우 이벤트 처리하는 방법을 이해한다. GLFW에서 윈도우 이벤트를 처리를 위해 콜백함수(callback function)를 등록하는 방법을 학습한다. 사용자 입력에 반응하는 그래픽스 프로그램을 작성하는 기법을 익힌다. 사용자 입력을 통해 물체의 색깔이나 위치를 바꾸는 방법을 이해한다.

Index Terms

사용자 입력 처리, 콜백함수(callback function), 키보드 입력 처리, 모델 행렬(model matrix) 이동변환(translation)

1 사용자 입력에 반응하는 프로그래밍

일반적으로 컴퓨터그래픽스 프로그램은 사용자의 키보드 혹은 마우스 입력에 반응하여 화면이 업데이트 된다. 예를 들어 FPS 게임의 경우 사용자가 키보드를 두르면 캐릭터 위치가 바뀌고 마우스를 움직이면 시선 방향이 바뀌도록 설계되어 있다. 따라서, 캐릭터의 바뀐 위치와 시선 방향을 고려하여 화면을 새롭게 업데이트 해야 한다.

여기서는 사용자 입력에 반응하는 간단한 형태의 모던 OpenGL 프로그램을 작성한다. 이를 통해 다음을 학습한다.

- 키보드 입력을 받아 물체의 이동변환을 설정하는 방법을 학습한다.
- 물체가 해당 위치로 이동할 수 있도록 모델 행렬(model matrix)를 구성하는 방법을 학습한다.
- 구성된 모델 행렬을 이용해 어떤 방식으로 화면 구성이 되는지 학습한다.
- 물체의 이동을 이용해 간단한 애니메이션을 작성한다.

1.1 소스코드 컴파일 및 실행

터미널에서 git 명령어로 소스코드를 다운로드 받는다.

```
$ git pull https://github.com/kmuvcl/2021_graphics
```

다운로드 받은 소스코드 디렉토리로 이동한 후, 솔류션 코드를 아래와 같이 `make` 명령어를 통해 컴파일을 수행하자. 컴파일이 성공적으로 끝나면 현재 디렉토리에 실행가능한 파일인 `interaction`이 생성된다. 터미널에서 `interaction`을 실행시켜 빨간색 삼각형이 뜨는 OpenGL 프로그램이 구동되는지 확인한다.

```
$ cd 2021_graphics/tutorial/02.User_Interactions/cpp
$ cp solution/main.cpp .
$ make
$ ./interaction &
```

2 무작정 해보기

실행한 OpenGL 프로그램이 아래의 키보드 입력에 반응하여 업데이트되는지 살펴보자.

- `=/+` 키: 배경색 밝게 하기
- `-/_` 키: 배경색 어둡게 하기
- `w` 키: 물체 와이어프레임 렌더링 모드 on/off 토글
- `SPACE` 키: 물체 애니메이션 모드 on/off 토글
- `h/l` 키: 물체 위치 $\pm x$ 방향 이동
- `j/k` 키: 물체 위치 $\pm y$ 방향 이동

국민대학교 소프트웨어학부, junho@kookmin.ac.kr

3 실습 코드 작성해 보기

이제 뼈대 코드인 `skeleton/main.cpp`에서부터 차근차근 코드를 작성하여 동일한 기능을 가지는 프로그램을 완성해 보도록 하자.

```
$ cp skeleton/main.cpp .
$ vi main.cpp        # 자신이 익숙한 에디터로 main.cpp 코드를 수정한다.
```

실습 코드를 완성하는데 있어 핵심이 되는 사항은 다음 세 부분이다.

1) 콜백함수 선언 및 등록: 사용자 입력을 처리할 콜백 함수를 선언하고 등록한다. 한번 등록해 두면, GLFW는 이벤트가 발생될 때마다 등록해 놓은 콜백함수를 자동으로 호출한다.
2) 콜백함수 구현: 선언한 콜백 함수를 구현한다. 앞서, 우리가 정의한 변수는 콜백 함수를 구현한 코드 내에서 목적에 맞게 업데이트 한다.
3) 업데이트된 변수 적용: 콜백함수에 의해 업데이트된 변수를 OpenGL 함수에 넘겨 화면이 업데이트 되도록 한다.

이제 `main.cpp` 코드를 수정하여 우리가 정의한 특정 키가 눌리면 사용자 입력에 반응할 변수들을 업데이트하도록 하자. 먼저 OpenGL 프로그램에서 사용자의 입력에 반응할 변수를 소스코드 `main.cpp`에 앞쪽에 다음과 같이 정의한다.

```
// 배경색 관련 변수
float   g_bg_color_r = 0.5f, g_bg_color_g = 0.5f, g_bg_color_b = 0.5f;
// 와이어프레임 렌더링 모드 on/off 토글 변수
bool    g_is_wireframe_mode = false;
// 애니메이션 모드 on/off 토글 변수
bool    g_is_animaiton = false;
// 물체 위치 변수
float   g_translate_x = 0.0f, g_translate_y = 0.0f, g_translate_z = 0.0f;
```

3.1 키보드 이벤트 콜백함수 선언 및 등록

GLFW는 윈도우가 전달하는 각종 이벤트에 반응할 수 있도록 설계되어 있다. 여기서 이벤트라 함은 키보드 입력, 마우스 움직임 및 버튼 클릭, 화면의 크기 변경 등을 말한다.

기본적으로 GLFW로 작성된 프로그램은 각 이벤트에 대해 아무런 반응을 보이지 않는다. 만일 프로그래머가 특정 이벤트(예를 들어 키보드 누르기)에 대해 반응할 수 있도록 작성하려면 GLFW가 정의한 방식대로 콜백함수를 등록하고, 이벤트가 적절히 처리되도록 해당 콜백함수를 구현해야 한다. GLFW에서 키보드 입력을 다루기 위한 자세한 사항은 알기 위해서는 Keyboard Input 문서를 읽어보도록 하자.

여기서는 키보드 입력에 따라 반응하는 프로그램을 작성하기 때문에, 키보드 이벤트를 다룰 수 있는 콜백함수 `key_callback()`를 선언하고 `glfwSetKeyCallback()`로 콜백함수를 GLFW에 등록한다.

콜백함수를 등록할 때 유의해야할 점은 두가지는 아래와 같다.

1) 콜백함수의 입출력은 반드시 GLFW에서 정해놓은 형식을 따라야 한다. `glfwSetKeyCallback()`에 등록할 키보드 콜백함수는 `GLFWkeyfun`의 입출력 형식을 따라야 한다. 우리가 작성한 콜백함수 `key_callback()`는 `GLFWkeyfun`의 입출력 형식을 따르고 있음에 주목하자.
2) 콜백함수는 반드시 렌더링 루프 이전에 등록해야 한다. 우리가 작성한 콜백함수 `key_callback()`는 렌더링 루프에 들어가기 전에 `glfwSetKeyCallback()`를 이용해 GLFW에 등록하였음을 주목하자.

```
// 키보드 이벤트 콜백함수 선언
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods);

int main(void)
{
  // ...

  /// 키보드 콜백함수 등록 (* 반드시 렌더링 루프 이전에 등록)
  glfwSetKeyCallback(window, key_callback);

  // Loop until the user closes the window
  while (!glfwWindowShouldClose(window))
  {
    // ...
  }
  // ...
}
```

3.2 키보드 이벤트 콜백함수 구현

키보드 이벤트 처리를 담당할 콜백함수 `key_callback()` 함수를 다음과 같이 `main.cpp` 파일에 구현하여 키에 따라 변수가 적절히 업데이트 되도록 한다. GLFW에서 키보드의 각 키를 어떻게 정의해서 다루는지 자세히 알아보려면 Keyboard keys 문서를 읽어보도록 하자.

```
// 키보드 이벤트 콜백함수 구현
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
  // =/+ key: 배경색 점점 밝아지게
  if (key == GLFW_KEY_EQUAL && action == GLFW_PRESS)
  {
    g_bg_color_r += 0.1f;
    g_bg_color_g += 0.1f;
    g_bg_color_b += 0.1f;
    g_bg_color_r = std::min(1.0f, g_bg_color_r);
    g_bg_color_g = std::min(1.0f, g_bg_color_g);
    g_bg_color_b = std::min(1.0f, g_bg_color_b);
  }
  // -/_ key: 배경색 점점 어두워지게
  if (key == GLFW_KEY_MINUS && action == GLFW_PRESS)
  {
    g_bg_color_r -= 0.1f;
    g_bg_color_g -= 0.1f;
    g_bg_color_b -= 0.1f;
    g_bg_color_r = std::max(0.0f, g_bg_color_r);
    g_bg_color_g = std::max(0.0f, g_bg_color_g);
    g_bg_color_b = std::max(0.0f, g_bg_color_b);
  }

  // 와이어프레임 렌더링 모드 토글링
  if (key == GLFW_KEY_W && action == GLFW_PRESS)
    g_is_wireframe_mode = !g_is_wireframe_mode;

  // 와이어프레임 렌더링 모드 토글링
  if (key == GLFW_KEY_SPACE && action == GLFW_PRESS)
    g_is_animaiton = !g_is_animaiton;

  if (key == GLFW_KEY_H && action == GLFW_PRESS)     // move -x
    g_translate_x -= 0.1f;
  if (key == GLFW_KEY_L && action == GLFW_PRESS)     // move +x
    g_translate_x += 0.1f;
  if (key == GLFW_KEY_J && action == GLFW_PRESS)     // move -y
    g_translate_y -= 0.1f;
  if (key == GLFW_KEY_K && action == GLFW_PRESS)     // move +y
    g_translate_y += 0.1f;
}
```

3.3 업데이트된 변수 적용

이제 등록한 사용자 이벤트, 즉 키보드가 눌릴 때마다 키보드 콜백함수인 `key_callback()`가 호출된다. 또, `key_callback()` 함수에서 우리가 설계한 로직에 따라 특정 키가 눌리게 되면 그에 알맞게 변수가 업데이트 되었을 것이다.

이제 남은 일은 업데이트된 변수를 OpenGL에 적용하여 화면을 업데이트하는 것이다. 다음은 `main()` 함수에 정의된 루프로 GLFW에서 매 프레임마다 어떻게 OpenGL 렌더링을 처리할 것인지 정의한 코드 조각을 보여주고 있다.

```
int main(void)
{
  // ...

  // Loop until the user closes the window
  while (!glfwWindowShouldClose(window))
  {
```

```
    // Poll for and process events
    glfwPollEvents();

    // 사용자 정의 렌더링 코드 - BEGIN
    glClearColor(g_bg_color_r, g_bg_color_g, g_bg_color_b, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    set_transform();
    render_object();
    // 사용자 정의 렌더링 코드 - END

    // Swap front and back buffers
    glfwSwapBuffers(window);
  }

  // ...
}
```

매 프레임마다 어떤 방식으로 OpenGL 렌더링을 할 것인지는 사용자가 직접 정의할 수 있다. 위 코드 조각에서 사용자 정의 렌더링 코드는 크게 다음의 3가지 작업을 하고 있다.

1) 업데이트된 배경색 적용: `glClearColor()` 함수
2) 업데이트된 물체 변환 적용: `set_transform()` 함수
3) 업데이트된 렌더링 방식 적용: `render_object()` 함수

3.3.1 업데이트된 물체 변환 적용

OpenGL에서 배경색은 `glClearColor()`를 통해 설정할 수 있다. 다만, 배경색을 이용해 실제로 배경을 지우는 함수는 `glClear()`이므로, `glClear()`가 호출되기 전에 업데이트된 배경색을 `glClearColor()`를 통해 OpenGL에 전달해 주기만 하면 된다.

배경색은 매 프레임마다 렌더링 전에 이미지 전체에 적용해야 하므로, 위 코드조각처럼 렌더링 루프 안에서 정의된 사용자 렌더링 코드 중 첫번째로 불리는 코드가 되어야 한다.

3.3.2 업데이트된 물체 변환 적용

아래 코드조각은 변환(transformation)을 설정하는 `set_transform()` 함수이다. 여기서 `mat_view`와 `mat_proj`은 카메라의 외부/내부 파라메터와 관련된 행렬이고 현재 예제에서는 항등행렬로 설정하였다. 즉, 카메라에 대한 설정은 따로 설정하지 않은 것이다. 본 예제에서는 카메라의 변환을 다루지 않을 예정이니, 아래 코드조각에서 설정한 값을 그대로 사용하자.

물체의 변환은 모델 행렬 `mat_model`에서 다룬다. 이전 코드에서 키보드 입력에 따라 물체의 위치 이동을 `g_translate_x`, `g_translate_y`, `g_translate_z` 변수에 설정해 두었으므로, 여기서는 GLM 함수 `glm::translate`를 호출하여 이동 행렬(translation matrix)를 반환받고 이를 모델 행렬 `mat_model`로 설정한다.

또한, `g_is_animation`의 값이 `true`인 경우 애니메이션을 위해 물체의 x-축 이동 위치를 나타내는 값 `g_translate_x`를 업데이트 한다.

```
void set_transform()
{
  // set camera transformation
  mat_view = glm::mat4(1.0f);  // extrinsic param
  mat_proj = glm::mat4(1.0f);   // intrinsic param

  // TODO: 애니메이션 설정
  if (g_is_animation)
  {
    g_translate_x += 0.1f;
    if (g_translate_x > 1.0f)
      g_translate_x = -1.0f;
  }

  mat_model = glm::translate(glm::mat4(1.0f),
                             glm::vec3(g_translate_x, g_translate_y, g_translate_z));
}
```

3.3.3 업데이트된 렌더링 방식 적용

아래 코드조각은 업데이트된 렌더링 방식을 적용하여 물체를 렌더링하는 `render_object()` 함수이다. 앞서 키보드 입력을 처리하여 배경색과 와이어프레임 모드 여부에 관한 변수를 업데이트하였기 때문에 여기서는 업데이트된 변수를 이용해 물체를 렌더링하면 된다.

OpenGL에서 와이어프레임을 그리기 위해서는 여러가지 방법이 있지만, 가장 흔하게 사용되는 방식은 폴리곤의 내부를 채우지 않고 폴리곤의 경계인 라인(line) 부분만 그리게 하는 것이다. `glPolygonMode()`는 폴리곤을 채워서 그릴지 (`GL_FILL`), 그렇지 않고 경계만 그릴지 (`GL_LINE`)를 설정할 수 있는 함수이다.

```
// object rendering: 현재 scene은 삼각형 하나로 구성되어 있음.
void render_object()
{
  if (g_is_wireframe_mode)
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
  else
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

  // 쉐이더 프로그램 사용하여 렌더링 (이하 생략)
  // ...
}
```

4 컴파일 및 실행

완성된 코드를 컴파일하고 실행해 보고, 솔루션 코드와 동일하게 작동하는지 체크해 보도록 하자.

```
$ make
$ ./interaction &
```

5 연습문제

소스코드를 살펴보고 다음을 생각해 보자.

1) 물체를 회전시키거나 확대축소시키려면 어떻게 해야할까?
2) 여러 물체를 렌더링할 때는 어떻게 해야할까?
3) 여러 물체가 있을 때 물체마다 위치를 이동하거나 렌더링 방식을 달리 하려면 어떻게 해야할까?
4) `glPolygonMode()`에서 입력인자로 준 `GL_FRONT_AND_BACK`은 무엇을 의미하는지 조사해 보자.