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.

Index Terms

폴리곤, 삼각형, attribute 식별자, 모델 행렬, 뷰 행렬, 프로젝션 행렬, glVertexAttribPointer, glEnableVertexAttribArray, glDrawArrays, glDisableVertexAttribArray

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 이동 위치를 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/graphics
```

다운로드 받은 소스코드 디렉토리로 이동한 후, 터미널에서 make 명령어를 통해 컴파일을 수행하자. 컴파일이 성공적으로 끝나면 현재 디렉토리에 실행가능한 파일인 interaction이 생성된다.

```
$ make
```

터미널에서 아래와 같이 interaction을 실행시켜 빨간색 삼각형이 뜨는지 확인한다. 키보드에서 h/l 키을 누르면 삼각형이 좌우로 움직이는지 확인한다.

```
$ ./interaction &
```

2 무작정 해보기

2.1 키보드 입력을 이용하여 물체가 움직일 위치 설정

이제 삼각형을 위아래로도 움직이게 해 보자.

소스코드 중 main.cpp를 에디터로 열어 살펴보면, `key_callback()` 함수가 눈이 띌 것이다. 이 함수의 내용은 키보드 입력을 받아 전역 변수 `g_translate_x`, `g_translate_y`, `g_translate_z`의 값을 업데이트하는 것이다.

다음과 같이 j/k 키를 누르면 `g_translate_y` 값이 변경되도록 하자. 이제 j/k 키가 눌리면 삼각형이 위아래로 움직인다.

```
// 물체의 이동 변환 관련 변수
float   g_translate_x = 0.0, g_translate_y = 0.0, g_translate_z = 0.0;

void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
  if (key == GLFW_KEY_H)
    g_translate_x -= 0.1f;
```

국민대학교 소프트웨어학부, junho@kookmin.ac.kr

```
  if (key == GLFW_KEY_L)
    g_translate_x += 0.1f;

  // 위 아래로 움직이기 위해 추가한 부분
  if (key == GLFW_KEY_J)
    g_translate_y -= 0.1f;
  if (key == GLFW_KEY_K)
    g_translate_y += 0.1f;
}
```

2.2 키보드가 눌릴 때만 반응하도록 하기

삼각형이 이동되는 화면을 자세히 살펴보면 키보드를 한번 누를 때마다, 삼각형이 두번 이동하는 것을 관찰할 수 있다. 이는 현재의 코드가 사용자의 키보드 눌림 (press) 뿐만 아니라 키보드 때기(release) 이벤트에 둘 다 반응하기 때문이다.

다음과 같이 소스코드를 수정하여 키보드 눌림에만 반응하도록 하자.

```
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
  // action == GLFW_PRESS 조건을 추가
  if (key == GLFW_KEY_H && action == GLFW_PRESS)
    g_translate_x -= 0.1f;
  if (key == GLFW_KEY_L && action == GLFW_PRESS)
    g_translate_x += 0.1f;

  if (key == GLFW_KEY_J && action == GLFW_PRESS)
    g_translate_y -= 0.1f;
  if (key == GLFW_KEY_K && action == GLFW_PRESS)
    g_translate_y += 0.1f;
}
```

2.3 간단한 애니메이션 설정

이제 키보드를 누르지 않아도 삼각형이 왼쪽에서 오른쪽으로 움직이는 간단한 애니메이션을 만들어 보자.

다음과 같이 렌더링 루프에서 g_translate_x 값이 계속 변경되도록 하자.

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

  // Loop until the user closes the window
  while (!glfwWindowShouldClose(window))
  {
    // Poll for and process events
    glfwPollEvents();

    set_transform();
    render_object();

    // 루프가 돌면 x 값이 지속적으로 증가. (단, 1.0을 넘으면 다시 -1.0으로 설정)
    g_translate_x += 0.1f;
    if (g_translate_x > 1.0f)
      g_translate_x = -1.0f;

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

  // ...
}
```

2.4 정지/재시작이 가능한 애니메이션 만들기

키보드 p키가 눌리면 애니메이션이 정지하거나 재시작하는 기능을 추가해 보자. 우선, 애니메이션이 구동 중인지 아닌지를 표시하는 참거짓 변수 g_is_animation를 추가하자. 그런 다음 p키가 눌리면 g_is_animation의 참거짓 값이 토글(toggle)되도록 하자. 마지막으로 g_is_animation이 참일 때만 렌더링 루프에서 애니메이션이 실행되도록 하면 된다.

```
bool    g_is_animation = false;

void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
  // ...

  // p 키가 눌렸을 때, 정지/재시작 상태 토글
  if (key == GLFW_KEY_P && action == GLFW_PRESS)
  {
    g_is_animation = !g_is_animation;
    std::cout << (g_is_animation ? "animation" : "no animation") << std::endl;
  }
}

int main(void)
{
  // ...

    // g_is_animation가 참일 때만 애니메이션 수행
    if (g_is_animation)
    {
      // 루프가 돌면 x 값이 지속적으로 증가. (단, 1.0을 넘으면 다시 -1.0으로 설정)
      g_translate_x += 0.1f;
      if (g_translate_x > 1.0f)
        g_translate_x = -1.0f;
    }

  // ...
}
```

3 핵심코드 파악하기

본 예제에서 키보드 입력에 따라 물체를 이동시키기 위한 핵심코드는 main.cpp에서 아래 세 부분이다.

- 키보드 이벤트 처리를 위한 콜백함수(callback function)를 등록하고 구현하는 부분
- 물체가 특정 위치로 이동할 수 있도록 모델 행렬(model matrix)을 구성하는 부분
- 구성된 모델 행렬을 GPU에 반영하는 부분

3.1 키보드 이벤트 콜백함수

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 다음의 두 단계를 거쳐야 한다.

1) 해당 이벤트를 처리할 콜백함수 선언 및 등록
2) 콜백함수에서 응용프로그램의 목적에 맞게 이벤트가 처리되도록 함수 구현

키보드 입력에 따라 삼각형을 이동시켜야 하므로, 이번 예제에서는 다음과 같이 키보드 이벤트를 다룰 수 있는 GLFW콜백함수를 선언하고 등록하였다. 콜백 함수를 등록할 때 유의해야할 점은 반드시 렌더링 루프 이전에 콜백함수를 등록해야 한다는 것이다.

```
// 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))
  {
    // ...
  }
  // ...
}

// 키보드 이벤트 콜백함수 구현
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
  // 키보드 입력에 따른 업데이트
  // 이 예제에서는 g_translate_x, g_translate_y, g_translate_z 업데이트
}
```

이벤트 별로 어떤 콜백함수를 선언하고 등록해야 하는지는 GLFW레퍼런스를 참고하도록 한다.

콜백함수를 선언하고 등록하였으면, 이제 이벤트에 맞게 프로그램이 반응할 수 있는 논리를 코드로 구현하면 된다. 이 예제에서는 삼각형의 이동위치를 나타낼 `g_translate_x, g_translate_y, g_translate_z`의 값을 업데이트하였다.

3.2 모델 행렬 구성

이제 삼각형을 이동시키기 위한 모델 행렬이 어떻게 구성되었는지 살펴보도록 하자.

예제에서는 `set_transform()` 함수에서 삼각형의 이동위치 `g_translate_x, g_translate_y, g_translate_z`를 참고하여 다음과 같이 모델 행렬 `mat_model`을 구성한다.

```
kmuvcl::math::mat4x4f     mat_model, mat_view, mat_proj;

void set_transform()
{
  // set camera transformation
  mat_view.set_to_identity();   // extrinsic param
  mat_proj.set_to_identity();   // intrinsic param

  // set object transformation
  mat_model = kmuvcl::math::translate(g_translate_x, g_translate_y, g_translate_z);
}
```

모델 행렬 `mat_model` 이외, 다른 두 행렬 `mat_view, mat_proj`은 카메라의 외부/내부 파라메터와 관련된 행렬이고 현재 예제에서는 항등행렬로 설정하였다. 즉, 카메라에 대한 설정은 따로 설정하지 않은 것이다.

3.3 구성된 모델 행렬을 GPU에 반영

예제의 소스코드에 있는 `render_object()` 함수를 살펴보면 기존 예제에 있는 `render_object()` 함수와 거의 유사하지만 다음이 몇줄이 추가되어 있는 것을 확인할 수 있다.

```
kmuvcl::math::mat4x4f     mat_PVM;

void render_object()
{
  // ...

  // Setting Proj * View * Model
  mat_PVM = mat_proj * mat_view * mat_model;
  glUniformMatrix4fv(loc_u_PVM, 1, GL_FALSE, mat_PVM);

  // ...

}
```

코드 중, `mat_PVM = mat_proj * mat_view * mat_model` 부분은 정점 프로세서(vertex processor)가 처리하는 변환(transformation)에 관한 수식이다. 즉, 물체 좌표계(object-space)에 표현된 정점의 위치를 클립핑 좌표계(clipping-space)로 변환하는 수식을 나타낸다.

코드 중, `glUniformMatrix4fv(loc_u_PVM, 1, GL_FALSE, mat_PVM)` 부분은 메인 메모리에 있는 4x4 행렬 `mat_PVM`를 uniform 값의 형태로 GPU 메모리에 전달하는 부분이다. 지금 당장은 shader 코드를 이해할 필요가 없으므로, 메인 메모리에 있는 4x4 행렬이 GPU 메모리에 있는 4x4 행렬로 복사되었다고 생각하면 된다.

결국, 앞서 구성한 모델 행렬은 `m_PVM`에 곱해진 형태로 GPU에 반영된다.

3.4 유의 사항

반드시 유념해야할 사실은 렌더링 루프 안에서 모델 행렬을 설정하고 이를 GPU에 반영하도록 코드를 설계해야 한다는 것이다. 즉, 이 예제에서는 `set_transform()` 함수에서 모델 행렬을 설정하였고, `render_object()` 함수에서 모델 행렬을 `m_PVM`에 곱해진 형태로 GPU에 반영하였다. 따라서, 이 두 함수는 반드시 렌더링 루프 안에서 호출되어야 한다.

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

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

    /// (* 중요: 렌더링 루프 안에서 모델 행렬을 설정하고 이를 GPU 반영해야 함)
    set_transform();     // 모델 행렬 설정
    render_object();     // 모델 행렬을 GPU에 반영

    // ...
  }
  // ...
}
```

4 연습문제

소스코드를 살펴보고 다음을 생각해 보자.

1) 물체를 회전시키거나 확대축소시키려면 어떻게 해야할까?
2) 카메라의 이동과 투사변환을 나타내는 변수는 무엇인가?
3) 카메라는 전역 좌표계의 어디에 어떤 자세로 자리잡고 있을까? `mat_view`가 항등행렬로 설정된 것은 무엇을 의미하는가?
4) 카메라의 투사구조는 무엇인가? `mat_proj`가 항등행렬로 설정된 것은 무엇을 의미하는가?