Homework #1: 선형변환 함수 작성

Computer Graphics
School of Software, Kookmin University

Abstract

각종 선형변환(linear transformation)은 행렬을 이용하여 나타낼 수 있다. OpenGL 1.x와는 달리 OpenGL 2.x 이상의 모던 OpenGL은 컴퓨터 그래픽스 프로그래밍에 필요한 선형변환에 대한 행렬을 API 차원에서 더 이상 제공하지 않고 프로그래머가 직접 구현해서 쓰도록 강제하고 있다. 본 과제에서는 이동(translation)/회전(rotation)/확대축소(scaling) 등과 같은 표준변환 뿐만 아니라 카메라의 자세(camera pose)를 표현할 수 있는 시점변환(view transformation)과 카메라의 투영(camera projection) 형태를 표현할 수 있는 투영변환(projection transformation)을 4×4 행렬로 나타낼 수 있는 함수 작성을 목표로 한다.

1 과제 소개

선형변환(linear transformation)은 변환 전에 직선/평면 이었던 부분이 변환 이후에도 직선/평면으로 남아 있는 변환을 말한다. 모든 종류의 선형변환은 행렬(matrix)로 표현할 수 있다는 것이 수학적으로 알려져 있다.

컴퓨터 그래픽스 프로그래밍에서 자주 등장하는 표준변환인 이동(translation), 회전(rotation), 확대축소(scaling)은 모두 선형변환이며 각각 4×4 행렬로 표현할 수 있다. 뿐만 아니라, 카메라의 외부파라메터(extrinsic parameter)와 내부파라메터(intrinsic parameter)와 관련된 정보도 컴퓨터 그래픽스 프로그램에서는 4×4 행렬로 표현된다. 카메라의 외부파라메터와 관련있는 카메라 자세(camera pose) 정보는 4×4 시점변환(view transformation) 행렬로 표현된다. 카메라의 내부파라메터와 관련있는 카메라 투영(camera projection) 정보는 4×4 투영변환(projection transformation) 행렬로 표현된다.

기존의 OpenGL 1.x에서는 앞서 언급한 선형변환에 해당하는 4×4 행렬을 만들어주는 함수가 API 차원에서 제공되었다. 그러나 OpenGL 2.x 이상의 모던 OpenGL에서는 선형변환에 해당하는 행렬을 만드는 함수는 더 이상 제공하지 않고 그 구현을 프로그래머에게 맡겨두고 있다.

2 과제 목표

본 과제에서는 모던 OpenGL 프로그래밍에서 자주 활용되는 선형변환을 살펴보고 이에 해당하는 4×4 행렬을 반환하는 C++ 함수 작성을 목표로 한다. 본 과제의 구현은 새로운 소스코드 transform.hpp 파일을 만들고 그 안에 본 과제에서 앞으로 설명할 각종 선형변환 함수를 작성하도록 한다. 기존에 자신이 구현한 Homework #0 코드와 이번 과제에서 새롭게 작성한 transform.hpp를 합쳐 전체적인 프로그램이 정상적으로 동작하도록 한다. 프로그램의 정상작동 여부는 각종 선형변환 함수에 대해 간단한 파라미터를 넣어본 후 이에 맞는 행렬이 반환되는지를 화면에 찍어보는 방식으로 확인하도록 한다.

다음은 컴퓨터 그래픽스 프로그래밍에서 활용되는 각종 선형변환과 이에 대응하는 4×4 행렬을 소개하고 있다. 각 행렬을 반환하는 선형변환 함수를 설계하고 이를 transform.hpp 소스코드 내에 C++ 템플릿 함수 형태로 구현하는 것이 본 과제의 수행 내용이다.

이동변환 (translation) 행렬

x, y, z축으로 각각 d_x, d_y, d_z 만큼 이동하는 이동변환은 다음과 같은 4×4 행렬로 표현된다.

$$\begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

본 과제에서는 이동변환 행렬을 반환하는 다음과 같은 함수를 작성하도록 한다.

```
mat4x4f translate(float dx, float dy, float dz);
mat4x4d translate(double dx, double dy, double dz);
```

국민대학교 소프트웨어학부 김준호, junho@kookmin.ac.kr

회전변환 (rotation) 행렬

단위벡터 $\mathbf{u} = (u_x, u_y, u_z)$를 회전축으로 하여, 원점을 중심으로 θ 만큼 회전하는 회전변환은 다음과 같은 4×4 행렬로 표현된다.

$$\begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta & 0 \\ u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta & 0 \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

본 과제에서는 회전변환 행렬을 반환하는 다음과 같은 함수를 작성하도록 한다. 여기는 다음 사항에 유의하도록 한다.

- 함수입력으로 들어오는 변수 angle은 도(degree)로 표현된 각도이지만 Eq. (2)에서의 θ는 라디언(radian)으로 표현된 각도이다. 따라서, angle 값을 θ로 쓸 때 도를 라디언 값으로 변환시켜 사용해야 한다.
- 함수입력으로 들어오는 변수 x, y, z는 회전축을 지정한다는 의미를 가지기 때문에 단위벡터가 아닌 입력이 들어올 수 있다. 따라서, 입력으로 들어온 x, y, z를 이용하여 단위벡터 $\mathbf{u}$를 구한 다음 Eq. (2)에서의 u_x, u_y, u_z를 설정하도록 한다.

```
mat4x4f rotate(float angle, float x, float y, float z);
mat4x4d rotate(double angle, double x, double y, double z);
```

확대축소 (scaling) 행렬

x, y, z축으로 각각 s_x, s_y, s_z 만큼 원점을 기준으로 크기를 변화시키는 확대축소변환은 다음과 같은 4×4 행렬로 표현된다.

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

본 과제에서는 확대축소변환 행렬을 반환하는 다음과 같은 함수를 작성하도록 한다.

```
mat4x4f scale(float sx, float sy, float sz);
mat4x4d scale(double sx, double sy, double sz);
```

시점변환 (view transformation) 행렬

3차원에서 카메라의 자세와 관계된 시점변환은 다음과 같이 두 행렬의 곱으로 표현된 4×4 행렬로 표현된다.

$$\begin{bmatrix} \text{cam-x-axis}_x & \text{cam-x-axis}_y & \text{cam-x-axis}_z & 0 \\ \text{cam-y-axis}_x & \text{cam-y-axis}_y & \text{cam-y-axis}_z & 0 \\ \text{cam-z-axis}_x & \text{cam-z-axis}_y & \text{cam-z-axis}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\text{cam-pos}_x \\ 0 & 1 & 0 & -\text{cam-pos}_y \\ 0 & 0 & 1 & -\text{cam-pos}_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

본 과제에서는 시점변환 행렬을 반환하는 다음과 같은 함수를 작성하도록 한다. 여기는 다음 사항에 유의하도록 한다.

- 함수의 입력 (eyeX, eyeY, eyeZ)로부터 카메라 좌표계의 중심인 cam-pos는 설정한다.
- 함수의 입력 (centerX, centerY, centerZ)와 (eyeX, eyeY, eyeZ) 차를 구하고 단위벡터화(normalization) 한 후, cam-z-axis를 설정한다.
- 함수의 입력 (upX, upY, upZ)와 cam-z-axis의 외적(cross product)를 구하고 단위벡터화 한 후, cam-x-axis를 설정한다.
- cam-z-axis와 cam-x-axis의 외적를 구하고 단위벡터화 한 후, cam-y-axis를 설정한다.
- lookAt() 함수는 Eq. (4)에 나타난 두 개의 4×4 행렬에 대한 곱 연산결과인 4×4 행렬 하나를 반환하도록 한다.

```
mat4x4f lookAt(float eyeX, float eyeY, float eyeZ,
               float centerX, float centerY, float centerZ,
               float upX, float upY, float upZ);
mat4x4d lookAt(double eyeX, double eyeY, double eyeZ,
               double centerX, double centerY, double centerZ,
               double upX, double upY, double upZ);
```

투영변환 (projection transformation) 행렬

3차원에서 카메라의 투영변환은 직교투영(orthographic projection)인지 원근투영(perspective projection)인지에 따라 다음과 같은 4×4 행렬로 표현된다.

직교투영(orthographic projection) 행렬

3차원 카메라의 직교투영변환은 다음과 같은 4×4 행렬로 표현된다. 단, l, r, b, t, n, f는 각각 left, right, bottom, top, near, far의 약어이다.

$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

본 과제에서는 직교투영변환 행렬을 반환하는 다음과 같은 함수를 작성하도록 한다.

```
mat4x4f ortho(float left, float right,
              float bottom, float top,
              float nearVal, float farVal);
mat4x4d ortho(double left, double right,
              double bottom, double top,
              double nearVal, double farVal);
```

원근투영(perspective projection) 행렬

3차원 카메라의 원근투영변환은 다음과 같은 4×4 행렬로 표현된다. 단, l, r, b, t, n, f는 각각 left, right, bottom, top, near, far의 약어이다.

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{6}$$

본 과제에서는 원근투영 행렬을 반환하는 다음과 같은 함수를 작성하도록 한다. 여기는 다음 사항에 유의하도록 한다.

- perspective() 함수는 내부적으로 frustum() 함수를 호출하는 방식으로 구현한다.
- perspective() 함수의 입력 중 fovy는 수직 시야각(vertical field of view angle)으로 도(degree) 단위로 입력된다.
- perspective() 함수의 입력 fovy, aspect, zNear, zFar를 이용하여 frustum() 함수를 호출하는데 필요한 left, right, bottom, top, nearVal, farVal 값을 계산한다.

```
mat4x4f frustum(float left, float right,
                float bottom, float top,
                float nearVal, float farVal);
mat4x4d frustum(double left, double right,
                double bottom, double top,
                double nearVal, double farVal);

mat4x4f perspective(float fovy, float aspect, float zNear, float zFar);
mat4x4d perspective(double fovy, double aspect, double zNear, double zFar);
```

JavaScript를 이용하는 경우

본 교과과정이 진행되는 동안 웹브라우져에서 WebGL을 이용하여 과제와 프로젝트를 하려는 학생의 경우, JavaScript 언어 기반으로 구현하도록 한다. C++ 언어를 기반으로 제공된 뼈대코드와 함께, JavaScript 언어로 벡터와 행렬 기반 라이브러리를 제공하는 glm-js사이트를 참고하여 과제를 진행하도록 한다.

3 과제 제출방법 (매우 중요!!!)

- 본 과제는 개인과제이며, 각자 자신의 코드를 완성하도록 한다.
- 4월 10일(화) 23시 59분까지 가상대학에 업로드하도록 한다.
- 과제 코드는 **Ubuntu 16.04 LTS 환경에서 make 명령으로 컴파일 가능**하도록 작성한다.
- 과제 코드는 다음의 파일들은 하나의 압축파일로 묶어 **tar.gz** 파일 형식이나 표준 **zip** 파일 형식으로만 제출하도록 한다. 이때, 압축파일의 이름은 반드시 'OOOOOOOO_HW01.tar.gz (OOOOOOOO은 자신의 학번)'과 같이 자신의 학번이 드러나도록 제출한다.
  1) 소스코드 및 리소스 파일들
  2) Makefile
- 과제에 관한 질문은 오피스아워를 활용하도록 한다. 교육조교(teaching assistant, TA)에게 메일로 약속시간을 정한 후, 교육조교가 있는 연구실로 방문하여 물어보는 것도 매우 권장하는 방법이다.