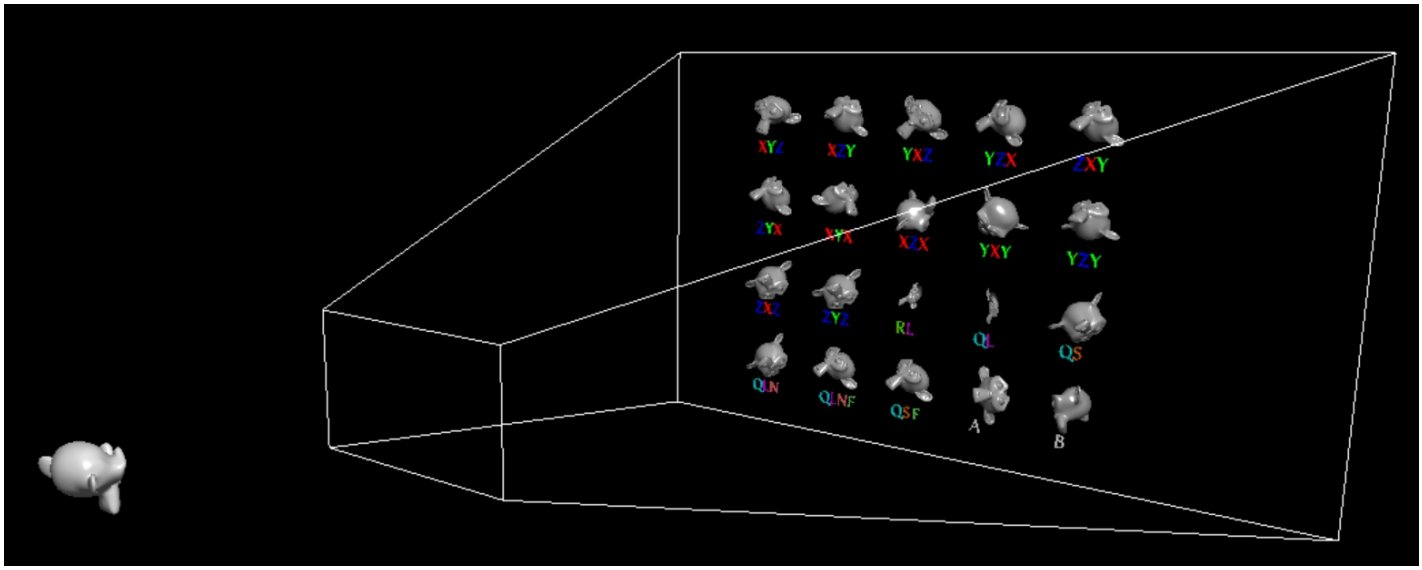
Assignment 2 - Lighting, Rotation Interpolation, Viewing, and Projection

In this assignment you will implement a GLSL lighting program, explore different methods for rotation interpolation, and use modeling, viewing, and projection transformations to draw a scene in OpenGL, including a visualization of the perspective projection frustum from a separate viewpoint.



Getting Started

We will be using different python modules than those we used in the first assignment. We will be using the `moderngl` module for bindings to the OpenGL 3.3+ core functionality, and the corresponding window creation tools from `moderngl-window`. We will use `scipy` for computing Euler angles from rotation matrices, and `pyrr` for some graphics related 3D math functions built on top of `numpy`:

```
python -m pip install moderngl
python -m pip install moderngl-window
python -m pip install scipy
python -m pip install pyrr
```

You may find the docs useful for these different libraries

- moderngl
- moderngl-window
- scipy as_euler function
- pyrr

A word of warning when providing matrices to OpenGL is that OpenGL expects data to be 32 bit floats ('f4') and the data must be in column-major form. Thus, a numpy matrix `M` should be transposed and flattened before writing it to a uniform, e.g., `M.T.flatten()`. In contrast, the `pyrr` library is built using `numpy` and `matrix44` class defined in this module is effectively a transposed version of the a `numpy` matrix to be more directly compatible with OpenGL. As

such, if you compose two `matrix44` matrices, be careful of the order! That is, using `A@B` with `pyrr` matrices gives the same result as first transforming by `A` and then transforming by `B`.

Provided Code

The provided code sets you up with a boilerplate framework for using the `moderngl` OpenGL module to draw to a window. It loads geometry from data files for a monkey head, and different letters, and also provides functions for setting up and drawing a VAO for a wire cube of size 2 centered at the origin. You can ignore the warning messages about "Unimplemented OBJ format statements" that are printed to the console.

In the code you will also see that it sets up rendering of a grid of objects created from the provided `Body` class to manage drawing a monkey head with different interpolations (you will need to expand this class to complete the assignment). The main render call will call the body `render` method with a varying interpolation parameter that over 4 seconds goes from 0 to 1, holds at 1 for one second, then goes from 1 to 0, and then holds at 0 for one second. With this repeating cycles, you will be able to examine an interpolation as it goes back and forth between two target orientations, `A` and `B`. The provided code also sets up the following keybindings that will let you (and the TA) interact with your solution.

Key	Description
a	Generates a random unit quaternion for target A
b	Generates a random unit quaternion for target B
z	Sets target B to be the negative quaternion of target A composed with a small random orientation change of 3 degrees
x	Sets target B to be the target A composed with a random orientation change of 180 degrees
i	Sets both targets to be the identity with quaternion (1,0,0,0)
1	Switches view and projection to the standard (first) camera view.
2	Switches view and projection to an alternate view (see teaser preview images at the top of the assignment), which allows us to see the viewer and projection frustum of the first view.

The provide code defines `rotation_type` which contains a list of 20 different short strings to define how the 20 different bodies in the grid should interpolate rotations. The first 12 are Euler angle orders, meaning the body should convert provided target orientations to Euler angles of the specified order and interpolate the angles to produce orientations that interpolate the targets. The remaining 8 rotation types are defined by the following table :

Type	Description
RL	Convert provided unit quaternion targets to 3x3 rotation matrices and interpolate all entries in the matrix linearly. You will note that the interpolated results will not be rotation matrices.
QL	Interpolate the two quaternions linearly. You will note taht the interpolated results will not be unit quaternions.
QS	Use a spherical linear interpolation of the two quaternions.
QLN	Interpolate the two quaternions linearly, and normalize the resulting vector so that it is unit length.
QLNF	Interpolate the two quaternions linearly, and normalize the result, but flip the sign of one of the quaternions if they point in opposite directions (i.e., F for 'fix')
QSF	Use spherical linear interpolation for the two quaternions, but flip the sign of one of the quaternions if they point in opposite directions.
A	Do not interpolate and only use orientation A
B	Do not interpolate and only use orientation B

See that there are some helper functions also provided for working with quaternions. While we saw formulas for quaternion multiplication and conversion of a quaternion to a rotation matrix in class, you may not have previously seen code for uniform sampling of a unit length vector. This is accomplished by normalizing a vector created with normal Gaussian

Objectives

This assignment has several steps where the marks associated with different steps are below.

1. **(2 marks) GLSL lighting program.** You are provided with a minimal GLSL program that transforms geometry and sets pixels to a constant colour. Implement an ambient plus Lambertian plus Blinn-Phong lighting model.

    i. You should take the light position and diffuse material colour as `uniform` inputs to your shaders, and set these uniforms from your python code. Use an ambient intensity of 0.3, a light position of (10,30,20). Note that you should use a diffuse material colour of (0.5,0.5,0.5) when drawing monkey heads. You will use colours defined in the `letter_colors` dict when drawing letter geometries.

    ii. Your lighting program should have a boolean `uniform` parameter to specify if lighting computations are to be performed or not. If drawing with lighting disabled, you should use the material color as the output color for all fragments processed. You will only use this mode for drawing the wire frame frustum.

2. **(2 marks) Draw body labels.** The grid spacing for the bodies is set up with a `pos` provided for each body to use as a translation in a modeling transformation. The provided code also loads geometry for letters for you to draw the rotation type labels for each body.

    - The origin of the object frame for each letter is at the center of the letter.
    - Each letter lies in the x-z plane, and has a small thickness in the y direction, and will need to have a rotation applied to lie in the x-y plane so as to be seen when viewing in the negative z direction.
    - The width of each letter is generally under 0.5 units across in x direction, making 0.5 units a good translation for laying out a sequence of letters.

- Draw the letter sequence centered 1.5 units below the monkey head.
- Use the colours defined in the `letter_colors` dict for drawing each letter of label.

3. **(3 marks) rotation interpolations**. The grid of bodies displayed on the screen is to visualize different interpolation methods described above. Use the interpolated rotation to draw the monkey head, but do not apply this rotation to the labels you draw for each body.

   i. Implement interpolation of Euler angles. Use the scipy `as_euler` function to compute the angles, and the corresponding `from_euler` and `as_matrix` functions to compute the 3x3 rotation matrix corresponding to a given set of interpolated Euler angles. Note that you may want to first use the provided `quatrion_to_R` function to make a rotation matrix from the unit quaternion before computing the Euler angles.

   ii. Implement the other rotation modes, as described in the table above.

4. **(2 marks) Draw viewer and frustum**. When the camera is changed to second view, you should be able to see a monkey model drawn at the origin of the first camera, looking towards the scene, and a wire-frame visualization of the frustum.

   i. Draw the viewer using the inverse view transform as a modelling transformation, and make that viewer a monkey head. You should also need to include a modeling transformation to rotate the monkey head such that it is looking towards the scene (i.e., down the negative z axis).

   ii. Draw the frustum using the inverse view transform and inverse projection transformation as modeling transformations on a wire cube. Mapping the -1 to 1 wire cube from the canonical viewing volume back to world coordinates is a handy way to draw the perspective projection frustum! Use the uniform you defined in the first step to disable the lighting when you draw the frustum, and use (1,1,1) as the colour for drawing, i.e., white.

5. **(1 mark) Answer Questions.** Provide answers to the following questions in your readme file.

   i. See that pressing 1 and 2 on the keyboard produces an interpolation between two different view and projections. How is it different from the other interpolations you are doing in the assignment, and what would you call this kind of interpolation?
   ii. Can problems arise when interpolating the viewing transformation matrices as done in the provided code?
   iii. Can problems arise when interpolating the projection transformation matrices as done in the provided code?
   iv. Think about why the QLN goes slow-fast-slow when interpolating almost opposite quaternions (the case generated by pressing 'z'). Can you see see the speed changes for 180 degree rotations (the case generated by pressing 'x')?
   v. Provided code converts a unit-length quaternion to a rotation matrix. What can you say about the matrices returned by this code when it is provided with a non-unit-length quaternion?

Finished?

Great! Submit your python implementation and updated vertex and fragment shader code in a zip folder, respecting the directory structure of the provided code. Use the comments at the top of each file you submit to list your name, student number. Put any other information you wish in a readme file (e.g., request to use your late penalty waiver). Do not submit an archive file, that is, no zip files for this assignment!

Note that you are encouraged to discuss assignments with your classmates, but not to the point of sharing code and answers. All code and written answers must be your own. Please see the course outline and the fine print in the slides of the first lecture.