CS 300 | Advanced Computer Graphics I
Assignment 1 | Phong Illumination Model

Description

In this assignment, the main task to apply the Phong illumination model on the shapes created for assignment 0. The parsed scene format will have all the content used in assignment 0 with new added data listed below:

- Lights in scene:
  - Every light will start with the keyword `light`.
  - All properties read after an object will be applied to the last object/light added:
    - `translation`: Translation of the light
    - `color`: light color.
    - `ambient`: ambient coefficient for the light.
    - `lightType`: Scale of the object
      - `POINT`: point light.
      - `SPOT`: spotlight.
      - `DIR`: directional light.
    - `spotAttenuation`: spotlight attenuation data with inner, outer angles and falloff of the transition from inner to outer angle for the spotlight. (spotlight only)
    - `attenuation`: a 3d vector containing the c_1, c_2 and c_3 coefficients for the attenuation formula. (point and spotlight only)
    - `direction`: a 3d vector containing the direction of the light. (directional and spotlight only)
- Material data:
  - Objects will have an extra parameter for the material:
    - `shininess`: controls how the specular light shines on the object. Default is 10.
- Animations are small updating behaviors that can be added to any object or light. Each element with an animation should call the corresponding update function provided in the `animations.h/.cpp` files and get the position of that element for the current frame. Along with the name of the animation a 3d vector will be loaded that serves as the parameter of the animation. Your framework will need to call the `Update` function every frame passing the current position and the time and it will get the new position that should be used for the render (warning: do not override the original position, keep a copy and that should be the input every frame). These animations can be concatenated, so if more than one is applied the output from an update should be the input to the next animation update.
  - `orbit`: will make the object/light orbit respect to a given center starting at its original position.
  - `sinusoidal`: will make the object/light oscillate up and down using the input parameter. Parameter are the phase of the wave, its frequency and the amplitude.

The application should:

1. Read the input file
   - a Load/Generate mesh data
   - b Generate Texture

2. Upload data to OpenGL
3. Accept from the keyboard the user input in order to move the camera
4. Update objects/light with the corresponding animation
5. Build the camera and perspective transformation according to the user input
6. Set shader program and uniforms
7. Draw each mesh
    - a **Vertex Shader**:
        - i Transform the geometry vertices from the model space to the homogenous clip space.
        - ii Calculate all required out variables for the lighting calculation and pass them to the fragment shader.
    - b **Fragment Shader**:
        - i Set up the texture samplers for diffuse and specular components.
        - ii Sample the appropriate texture and use it as the main diffuse color.
        - iii Set up the uniforms for the application to pass in lighting properties and material information.
        - iv Implement the Phong lighting equations explained in class.
8. Finish frame

Material Properties

- Most material properties are hardcoded with the following values:
    - Texture to map on shape (when texturing is disabled use UV as color)
    - Diffuse color (texture color)
    - Specular color (always white)
    - Shininess read from the scene file
    - Ambient coefficient will be 1

Light

- Requirement:
    - One light properly lighting the scene. If the scene file contains multiple lights and they are not supported use the first one loaded.
    - Extra credit will be rewarded if multiple lights are supported up to a maximum of 8 lights. If the scene file contains more than 8 lights keep only the first 8 listed.
- Properties:
    - Type of light
    - Its position and/or direction.
    - Light color. Specular component will be white.
    - Attenuation coefficients (for the spotlights this includes the angular attenuation parameters).

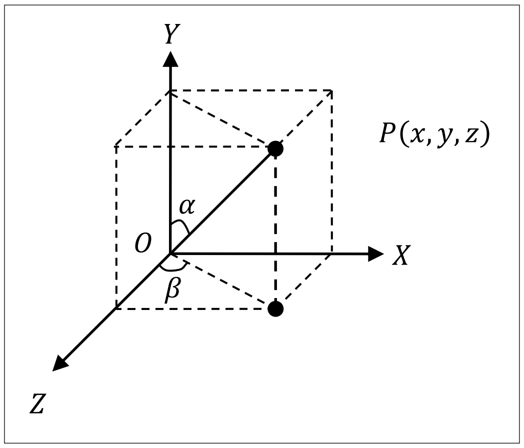
Shader Management

- Shader code MUST be loaded from an external file, i.e. you must NOT embed the shader code within your C++ code.
- Compile and install the shader code to be used. Should the shader/program fail to compile or link, you must print out the error messages to the console or the main window.

- The error message MUST specify which shader file(s) is causing the problem. (**Hint**: implement a *Shader Manager* class that encapsulates the one or more program objects – a program object contains a vertex and a fragment shader; also think that you might want to support more than one shader program in your application).

Input

- Camera will be controlled using spherical coordinates defined in the diagram. The target will be the origin of that coordinate system. The input should alter r, α and β as follows:
  - W: Make α angle smaller. It should move the camera towards the top of the target.
  - S: Make α angle greater. It should move the camera towards the bottom of the target.
  - A: Make β angle smaller. It should move the camera towards the left rotating around the target.
  - D: Make β angle greater. It should move the camera towards the left rotating around the target.
  - E: Make r greater. It should move the camera away from the target.
  - Q: Make r smaller. It should move the camera closer to the target.



- N: Toggle normal rendering
- T: Toggle texture-mapping on/off
- F: Toggle face/averaged normal
- M: Toggle wireframe on/off
- +/-: Increase/reduce number of slices (4 is the minimum number of slices)

Assignment Submission

Please refer to the syllabus for assignment submission guideline. Failure to the submission guidelines correctly might cause you to lose point.

Grading Rubrics

The following is a rough guideline on how your assignment will be graded and the weight of each part.

Feature	Grade %
Shaders	15%
Material	10%
Lighting	65%
Code quality	10%
Multiple Lights (EXTRA)	15%