Summer 2020
CS 300 | Advanced Computer Graphics I
Assignment 3 | Shadow Mapping with PCF

Description

For this programming assignment you need to implement shadow mapping with percentage closer filtering for a spotlight:

- **Shadow Map**
  This assignment will need two different passes in order to implement the shadow mapping technique. Each pass will require rendering the whole scene from different views:

  1. First pass will render the scene from the light's point of view. Remember to adjust the FOV so that it matches the spotlight angles.
  2. Second pass will render the scene from the cameras point of view and will use the shadow map generated in the previous pass to create the shadows.

- **Shader programs**

  - **Vertex Shader Shadow Map Generation**
    1. Calculate the vertex position using light as camera.

  - **Fragment Shader Shadow Map Generation**
    1. Store the depth value on the depth buffer.

  - **Vertex Shader Shadow Map Usage**
    1. Transform the vertex and the normal to the corresponding space.

  - **Fragment Shader Shadow Map Usage**
    1. Compute the fragment position in light space and the texture coordinates to sample the shadow map. Compare the distances of the fragment position and shadow map and decide if the fragment is in shadow or not.
    2. Use multiple shadow map samples to create soft shadows using percentage closer filtering with uniform sampling.
    3. Apply shadow value to the Phong lighting equation.

- **Scene and Setup**
  - Render a 100x100 units 2D plane aligned with the XZ plane and 15 units under the origin, i.e. the center is at (0,−15,0) and normal is (0,1,0).
  - Render one shape with size 20x20x20 at a time (from the shape library), arrows keys will rotate that shape.
  - Repeat the center shape with two adjacent objects that move up and down along the Y axis. Positions of those two shapes will be (-20,0,0) and (20,0,0) and scale will be 5x5x5.

- Place the light on a circle centered at (0,0,0) with radius of 20 and have it rotate slowly around the Y axis. While rotating they should move up/down in a sinusoidal manner along the Y axis. The spotlight should always point the object in the center.
- Apply the shader to the 3 shapes and the plane.
- Render a small sphere (radius = 1) to represent where the light(s) is (do not apply the shader on these spheres, they should be always white).
- Normal mapping is **NOT** required for this assignment and it should be disabled.
- Multiple lights are **NOT** required for this assignment and it should be disabled.
- The shadow map generated in the first pass should be visible in a small viewport on screen (with proper contrast adjustments if the texture looks fully white).

- **Input Handling**
  - Move the camera around in always looking at the object.
    - W: Move up.
    - S: Move down.
    - A: Move left.
    - D: Move right.
    - E: Further from object.
    - Q: Closer to object.

  - Select shape to be rendered through the number keys.
    - Numbers 1 to 5: Change the shape to be rendered
      - 1: Plane
      - 2: Cube
      - 3: Cone
      - 4: Cylinder
      - 5: Sphere
    - +: Increase the shape subdivisions
    - -: Decrease the shape subdivisions

  - P: Toggle to pause/start the light animation.
  - O: Toggle to pause/start adjacent object animation.
  - N: Toggle normal rendering
  - F: Toggle face/averaged normal
  - M: Toggle wireframe mode on/off
  - Z/X: Increase/Decrease the size of the neighborhood

  - Object rotation for center shape.
    - Arrows Up/Down: Rotate the shape along Y-axis
    - Arrows Right/Left: Rotate the shape along X-axis

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.

- Shadow map generation 30%
- Shadow map usage 35%
- PCF 20%
- Scene and presentation 10%

Percetage Closer Filtering

There are several ways to implement this shadow-edge anti-aliasing technique. We are going to follow the simple implementation technique that uses uniform sampling. Uniform sampling means that the samples are equally spaced and all of the have the same value. In practice, this results in getting the shadow map value on a square are with a defined size (3x3, 5x5…) and averaging the visibility of each of those samples. Here you have an incomplete code snippet:

```
float getPCFShadow(int index, vec2 uv)
{
    vec2 texelOffset = 1.0 / textureSize(shadowMap, 0);
    int neighbor = 2; // For a 5x5 neighborhood

    for(int x = -neighbor; x <= neighbor; x++)
    {
        for(int y = -neighbor; y <= neighbor; y++)
        {
            float shadowDepth = texture(shadowMap, uv + texelOffset * vec2(x, y)).r;

            // Evaluate visibility for this sample
        }
    }

    return accumulatedVisibility / sampleCount;
}
```