COSC422 Advanced Computer Graphics

Programming Exercise 09

Impostors

The aim of this exercise is to provide a good understanding of the use of framebuffer objects and the applications of render-to-texture methods in image based rendering.

Impostors.cpp:

The program `Impostors.cpp` displays a teapot as shown in Fig. 1. The teapot model contains 10816 triangles (see `teapot.dat`)

Fig. 1.

We will use a render-to-texture method to display 10 copies of the teapot without proportionaltely increasing the polygon count. The textures will be displayed as sprites (impostors), and any rotation of the teapot will then be seen in all 10 copies simultaneously. For off-screen rendering of the teapot, we require a framebuffer object. A framebuffer object (FBO) is a container for render buffers and texture objects.

In the initialize() function, create a FBO using `glGenFramebuffers()` function and bind it to the OpenGL context using `glBindFramebuffer()` (See slide [5]-17).

The program already includes code for the creation of a texture object with name "renderTex". Attach this texture object to the FBO using function `glFramebufferTexture2D()` (Slide [5]-17). Specify the attachement point as GL_COLOR_ATTACHMENT0. For offline rendering to this texture object, we further require a depth buffer. For this, we will use a render buffer. Create a renderbuffer object as shown on slide [5]-16. The width and the height of the renderbuffer object must be the same as the texture object. Attach the renderbuffer to the FBO using `glFramebufferRenderbuffer()` (Slide [5]-17).

While rendering the teapot, the output of the fragment shader should be directed to the attachment point where the texture object is attached. This is done by specifiying the corresponding parameter in the drawBuffers array:

```
GLenum drawBuffers[] = {GL_COLOR_ATTACHMENT0};
glDrawBuffers(1, drawBuffers);
```

Include the above statements in the initialize() function and check FBO completeness using `glCheckFramebufferStatus()` function (slide [5]-17).

The index of the attachment parameter in the array (in the above case, 0) is included as the location index in the layout qualifier for the fragment shader's out variable.

We are now ready to render the teapot to the texture object. In the display function, simply bind the framebuffer object to the context before rendering the teapot. Note that the `glClear()` function must be called *after* binding the FBO. When you run the program, you should get a blank screen as the teapot is not rendered to the default frame buffer, but to a texture object.

We will render the contents of the texture object as point sprites. The program already includes the definition of a vertex array object with name vaoID, containing 10 points. Each of these points will be converted to a point sprite, and the contents of the texture object mapped to it. Include a second rendering pass in the display() function as below:

```
glUniform1i(passLoc, 1);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glBindVertexArray(vaoID);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glDrawArrays(GL_POINTS, 0, 10);
glFlush();
```

The above code displays 10 points on the default framebuffer. The uniform variable "pass" is assigned a value 1 to distinguish between the two rendering passes. Modify the vertex shader code such that it executes the existing code only if the value of "pass" is 0. If the value of "pass" is 1, it should execute the following code for displaying point sprites:

```
vec4 pos = mvpMatrixS * vec4(position, 1.0);
gl_PointSize = (1.0 - pos.z / pos.w) * 128.0;
gl_Position = pos;
```

Note that rotational transformations cannot be applied to sprites, and therefore a different model-view-projection matrix is used. This matrix is already defined in the .cpp file.

The fragment shader also should be modified to output the color value of the teapot (vColor) if "pass" has a value 0, and the sampled texture object's colour given by `texture(renderTex, gl_PointCoord)` if pass has a value 1.

The program will now display 10 point sprites with the output from the first pass (the teapot) mapped to each of them (Fig. 2). Each sprite is an impostor. Use a timer

function to continuously rotate the teapot. This rotation will be seen in all point sprites.

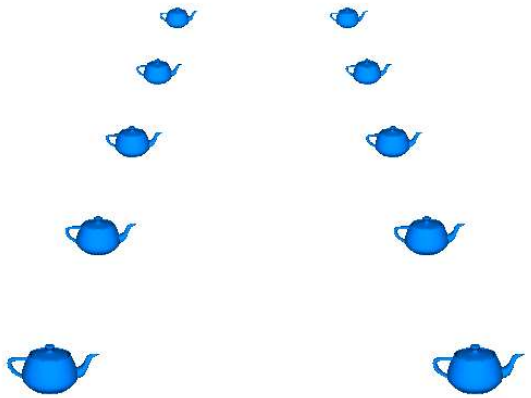

Fig. 2.

[5]: COSC422 Lecture slides “5 Image Based Rendering (IBR)”