COSC 4370 - Homework 4 Report
Name: Rosy Nguyen PSID: 2028823 November 2022

Goal: Implement texture mapping in OpenGL to reproduce a rotating textured cube. Write code to transfer the uv data to OpenGL buffer. Write the code for binding texture in the rendering loop and shader code to draw the texture.

- In order to map a texture to a cube we need to tell each vertex of the cube which part of the texture it corresponds to.
- Each vertex should thus have a texture coordinate associated with them that specifies what part of the TEXTURE IMAGE to sample from.
- Texture coordinates range from 0 to 1 in the x and y axis.
    - Texture coordinates start at (0,0) for the lower left corner to (1,1) for the upper right corner.
- Specify 4 texture coordinates points for the square.
    - The bottom-left side of the square correspond with the bottom-left side of the texture → (0,0)
    - The bottom-right side of the square correspond with the bottom-right side of the texture → (1,0)
    - The top-left side of the square correspond with the top-left side of the texture → (0,1)
    - The top-right side of the square correspond with the top-right side of the texture → (1,1)

UV COORDINATES
- U represents the horizontal axis (x-axis)
- V represents the vertical axis (y-axis)
- Each UV coordinate has a corresponding point in 3D space called a vertices.

FRAGMENT SHADER
- For each pixels in the fragment needs UV coordinates at which texture will be sampled.
- The sampler2D identifies which texture to access.
- In the glTextImage2D, the GL_RGB indicates that we are talking about a 3-component color: red, green, blue.
- Replace the original color assignment with the color of the texture at the specified UV
  Noted: need to convert to out vector 4.
  color = vec4(texture(myTextureSampler, UV).rgb, 1.0f)

VERTEX SHADER
- Generate the Model View Projection (MVP) matrix
  projection * view * model
- Set up the position of the vertex in clip space
  gl_Position = MVP matrix * vec4(position,1.0f)
- Notice that since the y coordinate tends to be flipped in images, we need to reverse the coordinate with
  UV = vec2(vertexUV.x, - vertexUV.y);

SET UP UV BUFFER
- Setting up UV buffer is similar to create vertex buffer
- Set the buffer container
  Gluint uv_ID;
- Generate buffer object name

glGenBuffers(number of buffer object name, the array in which the generated buffer object name are stored)

- Bind a named buffer object

  glBindBuffer(the target to which the buffer object is bound - in this case is the vertex attribute, name of a buffer object)

- Create and initialize the buffer object's data store

  glBufferData(the target to which the buffer object is bound - in this case is the vertex attribute, the size in bytes of the buffer object, a pointer to data that will be copied from, the expected usage pattern of the data store)

SET UP THE PROJECTION MATRIX

- Using the perspective projection for the scene.
- The four parameters:
  - The view angle in the y direction (45 degrees)
  - The aspect ratio of width to height (800f/600f)
  - The distance from the viewer (camera) to the near clipping plane (0.1f)
  - The distance from the viewer (camera) to the far clipping plane (100f)

SET UP THE VIEW MATRIX

- Using the lookat function to build the view.
- The three parameters:
  - Eye vector defines the position of the camera.
  - At vector defines the position where the camera (view) is (looking) pointing at.
  - Up vector defines the up direction of the camera.

BIND THE TEXTURE

- Generate buffer object name

  glGenBuffers(number of buffer object name, the array in which the generated buffer object name are stored)

- Bind the name buffer object

  glBindBuffer(the target to which the buffer object is bound - in this case is the vertex attribute, name of a buffer object)

- Create and fill the buffer object's data store

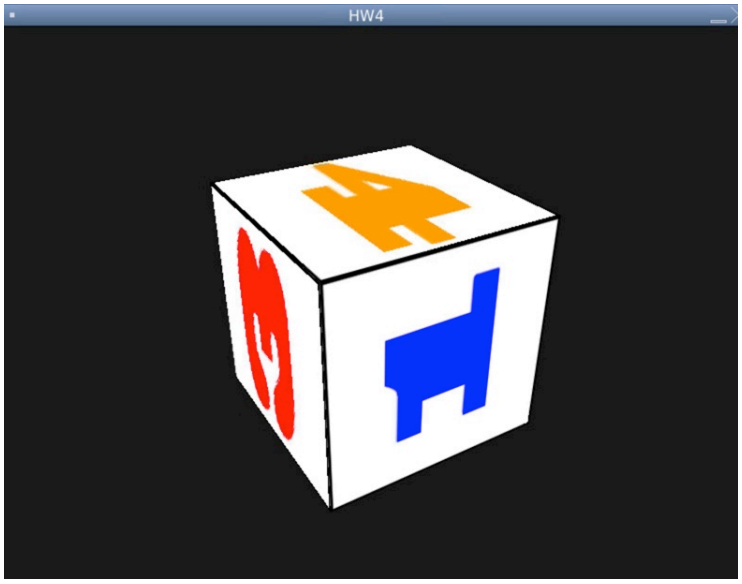
  glBufferData(the target to which the buffer object is bound - in this case is the vertex attribute, the size in bytes of the buffer object, a pointer to data that will be copied from, the expected usage pattern of the data store)

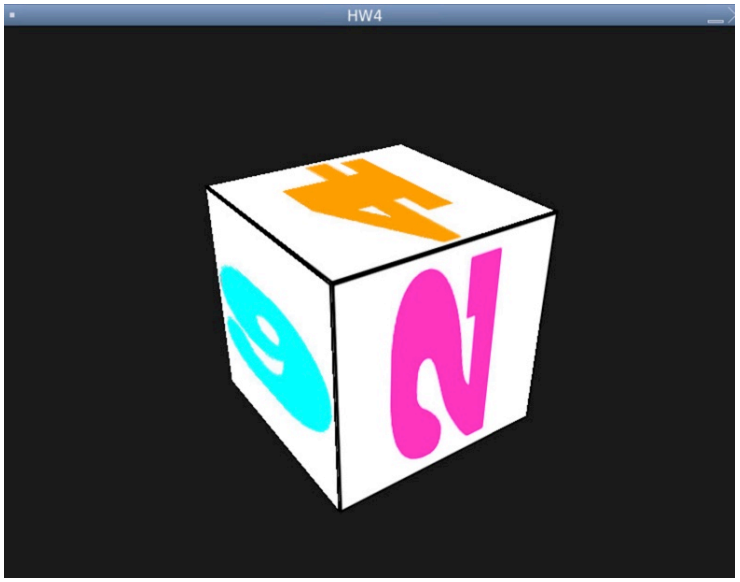
- Configure an array of generic vertex attribute data

  glVertexAttribPointer(the index of the generic vertex attribute to be modified, the number of components per generic vertex attribute, the data type of each component in the array, the fixed-point data values should not be normalized, the generic vertex attribute tightly packed in the array, offset of the first component of the first vertex attribute in the array)

- Enable the vertex attribute array

  glEnableVertexAttribArray(enable = 1)


HW4


HW4