Computer Graphics

12 - Lab - Texture Mapping

Yoonsang Lee
Hanyang University

Spring 2023

Correction for 8 - Lab- Lighting

- In 8 - Lab- Lighting, "Good Settings for Light & Material Phong Illumination Components" on page 16 should be modified as on the next page.

- The modified lecture materials were re-uploaded as "8 - Lab- Lighting-v2.pdf", and the corresponding lab code was also modified and pushed.

Good Settings for Light & Material Phong Illumination Components

- Light
  - diffuse, specular: color of the light source
  - ambient: the same color, but at much reduced intensity (about 10%)

- Material
  - diffuse, ambient: color of the object
  - specular:
    - ~~color of the light source~~ **white** (non-metal)
    - color of the object (metal)

Outline

- Install Pillow
- A simple texture mapped triangle
- Texture Filtering
- Mipmaps
- Texture Wrapping
- Multiple Textures

Install Pillow

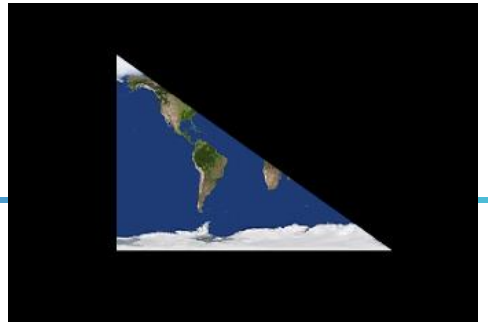
- Pillow: Popular python imaging library.

- Install:

```
$ workon cg-course
$ pip install pillow
```

- Documentation:
  - https://pillow.readthedocs.io/en/stable/index.html

[Code] 1-triangle-texture



- Let's start from "5-Lab-3DTransformations-VertProcess1/3-lookat.py".

- Remove code related to drawing a 'frame'.

- Import PIL (Pillow):

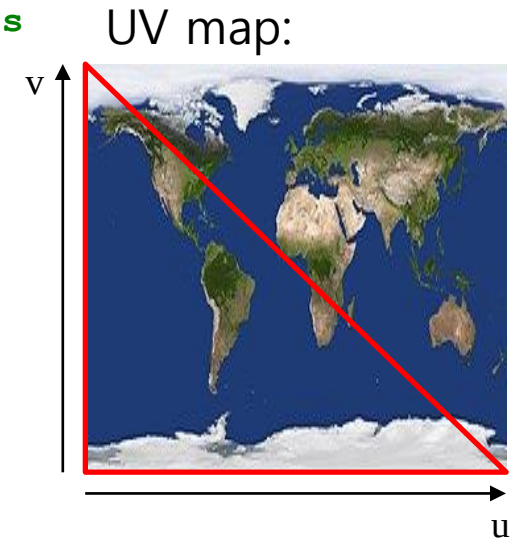
```
from PIL import Image
```

[Code] 1-triangle-texture

- VAO
  - Now our vertex data includes not only vertex positions and color values but also **texture coordinates**.

```
def prepare_vao_triangle():
    # prepare vertex data (in main memory)
    vertices = glm.array(glm.float32,
        # position      # color          # texture coordinates
         0.0, 0.0, 0.0,  1.0, 0.0, 0.0,  0.0, 0.0,  # v0
         0.5, 0.0, 0.0,  0.0, 1.0, 0.0,  1.0, 0.0,  # v1
         0.0, 0.5, 0.0,  0.0, 0.0, 1.0,  0.0, 1.0,  # v2
    )

    # create and activate VAO (vertex array object)
    ...
    # create and activate VBO (vertex buffer object)
    ...
    # copy vertex data to VBO
    ...
```



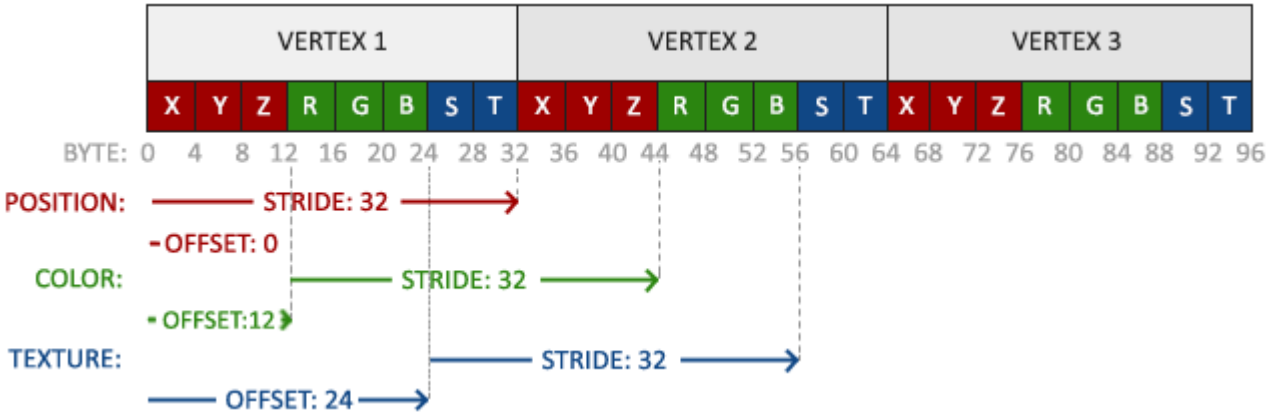
[Code] 1-triangle-texture

```
def prepare_vao_triangle():
    ...
    # configure vertex positions
    # glVertexAttribPointer(index, size, type, normalized, stride, pointer)
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * glm.sizeof(glm.float32),
None)
    glEnableVertexAttribArray(0)

    # configure vertex colors
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * glm.sizeof(glm.float32),
ctypes.c_void_p(3*glm.sizeof(glm.float32)))
    glEnableVertexAttribArray(1)

    # configure texture coordinates - 2D data
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * glm.sizeof(glm.float32),
ctypes.c_void_p(6*glm.sizeof(glm.float32)))
    glEnableVertexAttribArray(2)

    return VAO
```



* The image is from https://learnopengl.com/Getting-started/Textures

[Code] 1-triangle-texture

- Load an image and create a texture object
  - Create texture object and bind it - `glGenTextures`, `glBindTexture`
  - Load an image - using pillow (`Image.open`)
  - Specify a texture image using the loaded image - `glTexImage2D`

[Code] 1-triangle-texture

```
def main():
    ...
    # create texture object
    texture1 = glGenTextures(1)                 # create texture object
    glBindTexture(GL_TEXTURE_2D, texture1)      # activate texture1 as GL_TEXTURE_2D

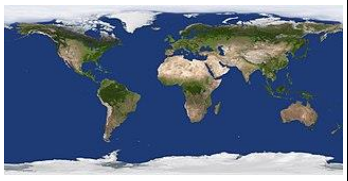
    # set texture filtering parameters - skip at this moment
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST)

    try:
        img = Image.open('./320px-Solarsystemscope_texture_8k_earth_daymap.jpg')

        # vertically filp the image
        # because OpenGL expects 0.0 on y-axis to be on the bottom edge, but images
usually have 0.0 at the top of the y-axis
        img = img.transpose(Image.FLIP_TOP_BOTTOM)

        # specify a 2D texture image
        # glTexImage2D(target, level, texture internal format, width, height,
border, image data format, image data type, data)
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, img.width, img.height, 0, GL_RGB,
GL_UNSIGNED_BYTE, img.tobytes())

        img.close()
    except:
        print("Failed to load texture")
```



[Code] 1-triangle-texture

- ```
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, img.width, img.height, 0,
  GL_RGB, GL_UNSIGNED_BYTE, img.tobytes())
  ```

- **glTexImage2D**(target, level, internalformat, width, height, border, format, type, data)
  - Specify a 2D texture image
  - target: Texture target. Set GL_TEXTURE_2D for 2D texture.
  - level: Level of detail. Set 0 (we'll use glGenerateMipmap() later).
  - internalformat: Format we want to store the texture.
  - width, height: Width / height of the image.
  - border: Must be 0 (some legacy stuff).
  - format, type: Format and datatype of source image.
  - data: Pointer to source image data in memory.
  ```

[Code] 1-triangle-texture

- Vertex shader

```
#version 330 core
layout (location = 0) in vec3 vin_pos;
layout (location = 1) in vec3 vin_color;
layout (location = 2) in vec2 vin_uv;

out vec4 vout_color;
out vec2 vout_uv;

uniform mat4 MVP;
void main()
{
    // 3D points in homogeneous coordinates
    vec4 p3D_in_hcoord = vec4(vin_pos.xyz, 1.0);

    gl_Position = MVP * p3D_in_hcoord;

    vout_color = vec4(vin_color, 1.);
    vout_uv = vin_uv;
}
```

[Code] 1-triangle-texture

- Fragment shader

```
#version 330 core
in vec4 vout_color;
in vec2 vout_uv;  // interpolated texture coordinates

out vec4 FragColor;

uniform sampler2D texture1;  // sampler2D: GLSL built-in datatype for 2D texture object

void main()
{
    //FragColor = vout_color;

    // vec4 texture(sampler, uv)
    // : retrive the color of the specified texture at the specified texture coordinates
    //   sampler: texture sampler2D
    //   uv: texture coordinates
    FragColor = texture(texture1, vout_uv);
}
```

[Code] 1-triangle-texture

- Drawing code is the same as the previous one.

```
def main():
    ...
    while not glfwWindowShouldClose(window):
        ...
        # draw triangle w.r.t. the current frame
        glBindVertexArray(vao_triangle)
        glDrawArrays(GL_TRIANGLES, 0, 3)
        ...
```

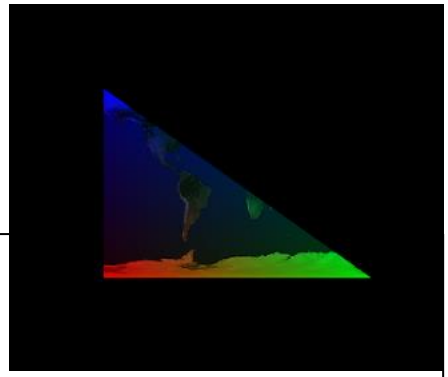
[Code] 1-triangle-texture

- Why don't we need to set the value of the uniform variable `texture1`?
  - By using `glUniform1i`, we can assign a location value to the texture sampler, to use multiple textures simultaneously in a fragment shader. This location is commonly referred to as a **texture unit**.
  - The **default active texture unit** is the **first texture unit indexed as 0**, and **`GL_TEXTURE0` is used as a reference to it**, enabling access and manipulation of the first texture unit.
  - Since GLSL initializes the value of a uniform variable to 0, if you are using only one texture, there is no need to explicitly set the value of sampler uniform variable to 0.
  - However, not all graphics drivers assign this default texture unit, so this code may not have rendered correctly for you.
    - In such a case, uncomment following lines in the code:

```
# glActiveTexture(GL_TEXTURE0)
# glBindTexture(GL_TEXTURE_2D, texture1)
```

[Code] 2-triangle-texture-color

- You can use the vertex colors as well, for example by multiplying a color value by a texture value.



- Fragment shaders

```
...
void main()
{
    ...
    FragColor = texture(texture1, vout_uv) * vout_color;
}
```

[Code] 3-triangle-texture-filter

- Let's draw a bigger triangle:

```
def main():
    ...
    while not glfwWindowShouldClose(window):
        ...
        # modeling matrix
        # M = glm.mat4()
        M = glm.scale(glm.vec3(5,5,5))
        ...
```

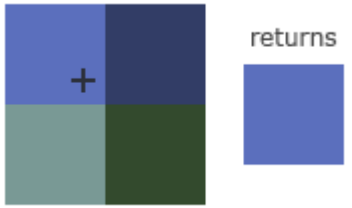
[Code] 3-triangle-texture-filter

- How can we get a "smoother" looking result?

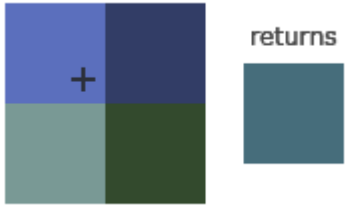
[Code] 3-triangle-texture-filter

- **Texture filtering** determines how to compute the color value corresponding to the given (u, v).
  - Texture coordinates (u, v) are given as floating-point values that do not precisely match the exact position of each pixel in the texture image.

- `GL_NEAREST`: OpenGL selects the *texel* (texture pixel) that center is closest to the texture coordinate.



- `GL_LINEAR`: OpenGL takes an interpolated value from the texture coordinate's neighboring texels.



* The images are from https://learnopengl.com/Getting-started/Textures

[Code] 3-triangle-texture-filter

```
def main():
    ...
    # create texture
    ...
    # set texture filtering parameters

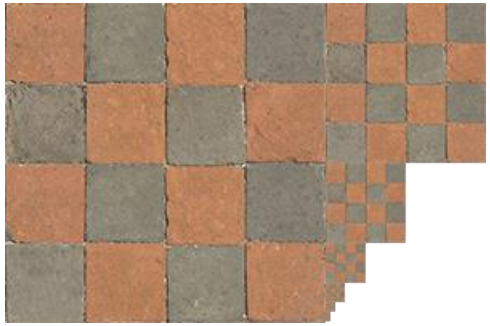
    # GL_TEXTURE_MIN_FILTER: used when the texture is displayed at a
smaller size than its original resolution.
    # default: GL_NEAREST_MIPMAP_LINEAR (will be expained soon)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)
    # glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)

    # GL_TEXTURE_MAG_FILTER: used when the texture is displayed at a
larger size than its original resolution.
    # default: GL_LINEAR
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST)
    # glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)

    try:
        img = Image.open('./320px-
Solarsystemscope_texture_8k_earth_daymap.jpg')
        ...
```

[Code] 4-triangle-texture-mipmaps

- Imagine a scene with very far objects mapped with high-resolution textures.
  - If this object is rendered by a single pixel, accessing the colors of all pixels in a high-resolution image would be very inefficient.



- That's why we use *mipmaps*.
- **Mipmaps** are a collection of texture images, where each subsequent texture is half the size of the previous one.
  - Beyond a certain distance from the viewer, OpenGL switches to a mipmap texture that best fits the object's distance.
  - Each mipmap level represents the texture at a different level of detail.

* The image is from https://learnopengl.com/Getting-started/Textures

[Code] 4-triangle-texture-mipmaps

```
def main():
    ...
    # GL_TEXTURE_MIN_FILTER: used when the texture is displayed at a smaller size
than its original resolution.
    # default: GL_NEAREST_MIPMAP_LINEAR
    # GL_NEAREST_MIPMAP_NEAREST: takes the nearest mipmap to match the pixel size
and uses nearest neighbor interpolation for texture sampling.
    # GL_LINEAR_MIPMAP_NEAREST: takes the nearest mipmap level and samples that
level using linear interpolation.
    # GL_NEAREST_MIPMAP_LINEAR: linearly interpolates between the two mipmaps that
most closely match the size of a pixel and samples the interpolated level via
nearest neighbor interpolation.
    # GL_LINEAR_MIPMAP_LINEAR: linearly interpolates between the two closest mipmaps
and samples the interpolated level via linear interpolation.
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_LINEAR)

    # GL_TEXTURE_MAG_FILTER: used when the texture is displayed at a larger size
than its original resolution.
    # default: GL_LINEAR
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
 try:
        ...
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, img.width, img.height, 0, GL_RGB,
GL_UNSIGNED_BYTE, img.tobytes())
        # generate mipmaps
        glGenerateMipmap(GL_TEXTURE_2D)
```

[Code] 5-triangle-texture-wrap

- What happens if we specify texture coordinates outside the range from (0,0) to (1,1)?

- There are four **texture wrapping** options:

  - GL_REPEAT: The default behavior for textures. Repeats the texture image.
  - GL_MIRRORED_REPEAT: Repeats but mirrors the image with each repeat.
  - GL_CLAMP_TO_EDGE: Clamps the coordinates between 0 and 1, resulting in a stretched edge pattern.
  - GL_CLAMP_TO_BORDER: Coordinates outside the range are now given a user-specified border color.

* The image is from https://learnopengl.com/Getting-started/Textures

[Code] 5-triangle-texture-wrap

```
def prepare_vao_triangle():
    # prepare vertex data (in main memory)
    vertices = glm.array(glm.float32,
        # position      # color          # texture coordinates
         0.0, 0.0, 0.0,  1.0, 0.0, 0.0,  -.5, -.5,  # v0
         0.5, 0.0, 0.0,  0.0, 1.0, 0.0,  2.0, -.5,  # v1
         0.0, 0.5, 0.0,  0.0, 0.0, 1.0,  -.5, 2.0,  # v2
    )
    ...
```

[Code] 5-triangle-texture-wrap

```
def main():
    ...
    # set the texture wrapping parameters
    # default: GL_REPEAT

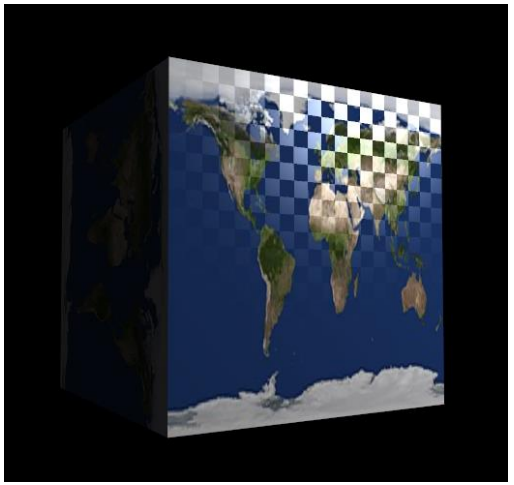
    # GL_TEXTURE_WRAP_S: in s-coordinate (== u-coordinate in uv space)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE)
    # glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER)
    # glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT)
    # glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
    # glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRROR_CLAMP_TO_EDGE)

    # GL_TEXTURE_WRAP_T: in t-coordinate (== v-coordinate in uv space)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE)
    # glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER)
    # glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT)
    # glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
    # glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRROR_CLAMP_TO_EDGE)
    try:
        ...
```

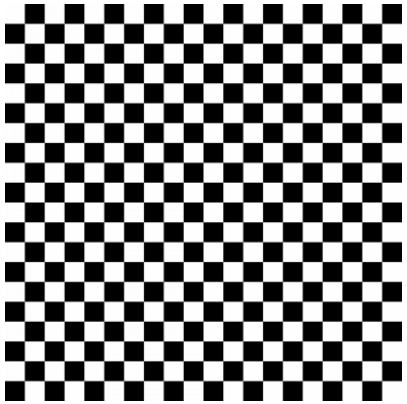
[Code] 6-cube-multiple-textures

- This example loads two different images and uses them as the diffuse and specular maps respectively.

diffuse map



result



specular map

[Code] 6-cube-multiple-textures

- Recall: By using `glUniform1i`, we can assign a location value to the texture sampler, to use multiple textures simultaneously in a fragment shader. This location is commonly referred to as a **texture unit**.

- To bind multiple textures simultaneously,
  - Assign texture units to samplers, by setting each sampler using `glUniform1i` (e.g., set 0 for the first texture unit).
  - Activate the desired texture unit using `glActiveTexture` (e.g., with GL_TEXTURE0 for the first texture unit).
  - Subsequent `glBindTexture` calls will then bind the texture to the active texture unit.

[Code] 6-cube-multiple-textures

- Let's start from "8-Lab-Lighting/4-all-components-phong-facenorm.py".
- Import PIL (Pillow).

- Add texture coordinates to the vertex data for VAO.

- Two texture sampler uniform variables: `texture_diffuse`, `texture_specular` connected to two texture objects with the same name.

- In the shaders, material color (diffuse & ambient color) comes from `texture_diffuse`, material specular color comes from `texture_specular`.

[Code] 6-cube-multiple-textures

```
def main():
    ...
    # diffuse texture
    texture_diffuse = glGenTextures(1)
    glBindTexture(GL_TEXTURE_2D, texture_diffuse)
    ...
    try:
        img = Image.open('./320px-Solarsystemscope_texture_8k_earth_daymap.jpg')
        ...
    # specular texture
    texture_specular = glGenTextures(1)
    glBindTexture(GL_TEXTURE_2D, texture_specular)
    ...
    try:
        img = Image.open('./plain-checkerboard.jpg')
        ...

    # for i-th texture unit, sampler uniform variable value should be i
    glUniform1i(glGetUniformLocation(shader_program, 'texture_diffuse'), 0)
    # activate i-th texture unit by passing GL_TEXTUREi
    glActiveTexture(GL_TEXTURE0)
    # texture object is binded on this activated texture unit
    glBindTexture(GL_TEXTURE_2D, texture_diffuse)

    glUniform1i(glGetUniformLocation(shader_program, 'texture_specular'), 1)
    glActiveTexture(GL_TEXTURE1)
    glBindTexture(GL_TEXTURE_2D, texture_specular)
```

[Code] 6-cube-multiple-textures

- Vertex shader

```
#version 330 core
layout (location = 0) in vec3 vin_pos;
layout (location = 1) in vec3 vin_normal;
layout (location = 2) in vec2 vin_uv;

out vec3 vout_surface_pos;
out vec3 vout_normal;
out vec2 vout_uv;

uniform mat4 MVP;
uniform mat4 M;
void main()
{
    vec4 p3D_in_hcoord = vec4(vin_pos.xyz, 1.0);
    gl_Position = MVP * p3D_in_hcoord;

    vout_surface_pos = vec3(M * vec4(vin_pos, 1));
    vout_normal = normalize( mat3(inverse(transpose(M)) ) * vin_normal);
    vout_uv = vin_uv;
}
```

[Code] 6-cube-multiple-textures

- Fragment shader

```
...
in vec2 vout_uv;  // interpolated texture coordinates
...
uniform sampler2D texture_diffuse;
uniform sampler2D texture_specular;
void main()
{
    ...
    //vec3 material_color = vec3(1,0,0);
    vec3 material_color = vec3(texture(texture_diffuse, vout_uv));

    // light components
    ...
    // material components
    vec3 material_ambient = material_color;
    vec3 material_diffuse = material_color;

    //vec3 material_specular = vec3(1,1,1);  // for non-metal material
    vec3 material_specular = vec3(texture(texture_specular, vout_uv));

    ...
    vec3 color = ambient + diffuse + specular;
    FragColor = vec4(color, 1.);
}
```

[Code] 6-cube-multiple-textures

```
def prepare_vao_cube():
    # prepare vertex data (in main memory)
    # 36 vertices for 12 triangles
    vertices = glm.array(glm.float32,
        # position      # normal  # texture coordinates
        -1 ,  1 ,  1 ,  0, 0, 1,  0.0, 1.0,  # v0
         1 , -1 ,  1 ,  0, 0, 1,  1.0, 0.0,  # v2
         1 ,  1 ,  1 ,  0, 0, 1,  1.0, 1.0,  # v1

        -1 ,  1 ,  1 ,  0, 0, 1,  0.0, 1.0,  # v0
        -1 , -1 ,  1 ,  0, 0, 1,  0.0, 0.0,  # v3
         1 , -1 ,  1 ,  0, 0, 1,  1.0, 0.0,  # v2

        -1 ,  1 , -1 ,  0, 0,-1,  0.0, 1.0,  # v4
         1 ,  1 , -1 ,  0, 0,-1,  1.0, 1.0,  # v5
         1 , -1 , -1 ,  0, 0,-1,  1.0, 0.0,  # v6

        -1 ,  1 , -1 ,  0, 0,-1,  0.0, 1.0,  # v4
         1 , -1 , -1 ,  0, 0,-1,  1.0, 0.0,  # v6
        -1 , -1 , -1 ,  0, 0,-1,  0.0, 0.0,  # v7

        -1 ,  1 ,  1 ,  0, 1, 0,  0.0, 1.0,  # v0
         1 ,  1 ,  1 ,  0, 1, 0,  1.0, 1.0,  # v1
         1 ,  1 , -1 ,  0, 1, 0,  1.0, 0.0,  # v5
    ...
```

Time for Assignment

- No 'time for assignment' today.

- Email your assignment source code and captured video to TA by June 4.
  - babap8514@gmail.com

- If you have any questions about the lecture or lab, please post them on the LMS Q&A board.