

---

# **Computer Graphics**

## **12 - Lab - Texture Mapping**

Yoonsang Lee  
Hanyang University

Spring 2026

# Outline

---

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

# Install Pillow

---

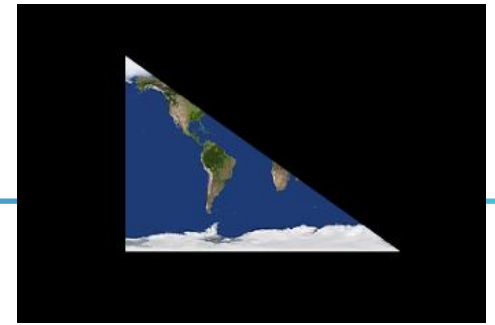
- Pillow: Popular python imaging library.

- Install:

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

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

# [Code] 1-triangle-texture



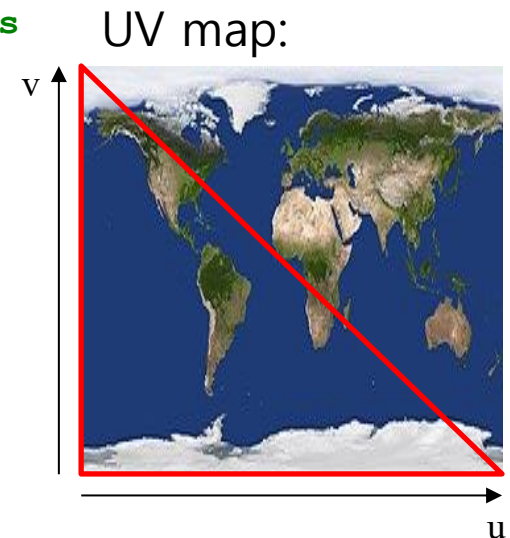
- Let's start from "5-Lab-3DTransformations-VertProcess1/3-lookat.py".
- Remove code related to drawing a 'frame'.
- Import PIL (Pillow):

```
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  
    ...
```



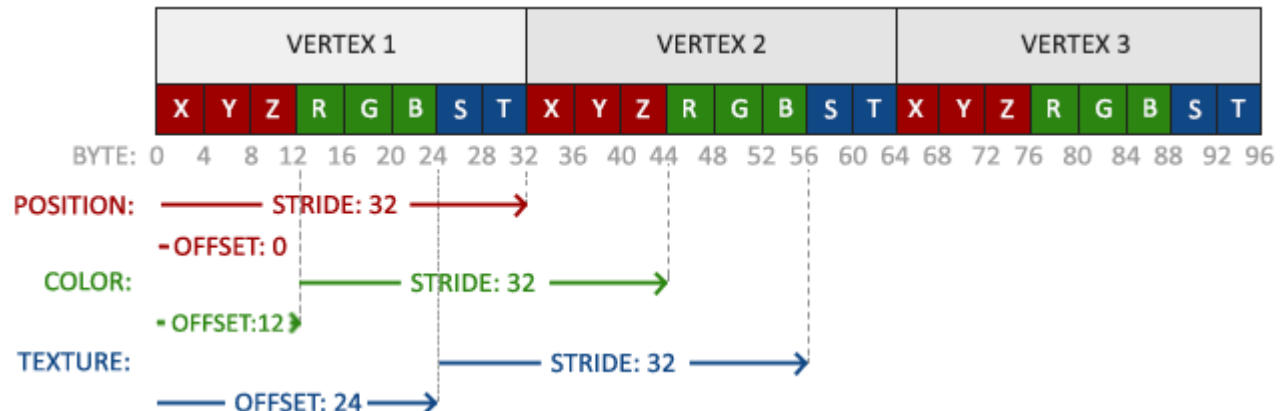
# [Code] 1-triangle-texture

```
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
```



# [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`)
  - Upload the image to the texture object - `glTexImage2D`
  - Configure texture sampling – `glActiveTexture`, `glUniform1i`

```

def main():
    ...
    # create texture object
    texture1 = glGenTextures(1)                # create texture object

    # use texture unit 0
    glActiveTexture(GL_TEXTURE0)
    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 flip 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)

        # upload image data to the currently bound texture object
        # 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")

    # connect sampler to texture unit 0
    loc_texture1 = glGetUniformLocation(shader_program, 'texture1')
    glUseProgram(shader_program) # updating uniform requires activating shader prog.
    glUniform1i(loc_texture1, 0)

```



# [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)
  - Upload image data to the currently bound texture object
  - 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. The sampler does not store a texture object ID. Instead,
it stores the texture unit index

void main()
{
    //FragColor = vout_color;

    // vec4 texture(sampler, uv)
    // : retrieve 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)  
        ...
```

# Texture Units

---

- A texture unit is a GPU texture slot that can hold a bound texture object.
- A sampler uniform references a texture unit rather than a texture object directly.
- `glUniform1i(loc_texture1, 0)`
- Here, **0** refers to Texture Unit **0** (`GL_TEXTURE0`).

# Texture Objects, Texture Units, and Samplers

- Texture Object:
  - image data
- Texture Unit
  - texture slot
- Sampler Uniform
  - selects a texture unit

- Relationship:

Texture Object

↓ `glBindTexture`

Texture Unit (e.g. `GL_TEXTURE0`)

↓ `glUniform1i(..., 0)`

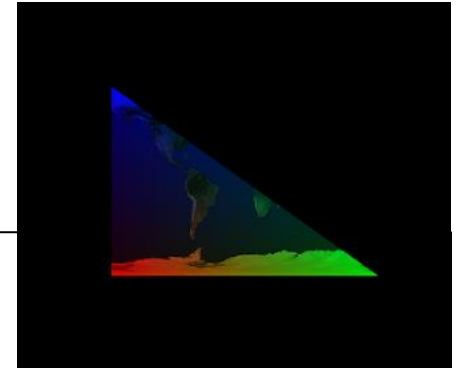
Sampler (`sampler2D texture1`)

# [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.



# [Code] 3-triangle-texture-filter

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

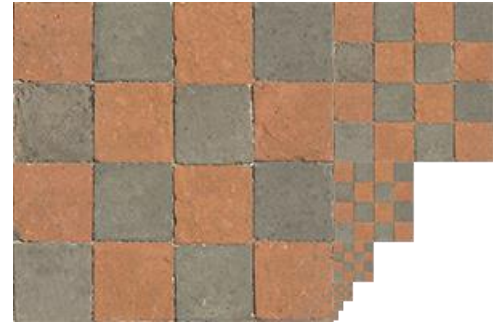
    # GL_TEXTURE_MIN_FILTER: used when the texture is displayed at a
    smaller size than its original resolution.
    # default: GL_NEAREST_MIPMAP_LINEAR (will be explained soon)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)
    # glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,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



# [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

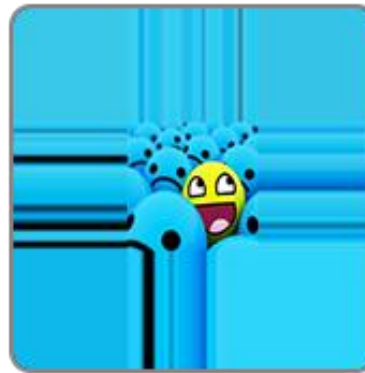
- What happens if we specify texture coordinates outside the range from (0,0) to (1,1)?
- There are four **texture wrapping** options:



GL\_REPEAT



GL\_MIRRORED\_REPEAT



GL\_CLAMP\_TO\_EDGE



GL\_CLAMP\_TO\_BORDER

- 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.

# [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)

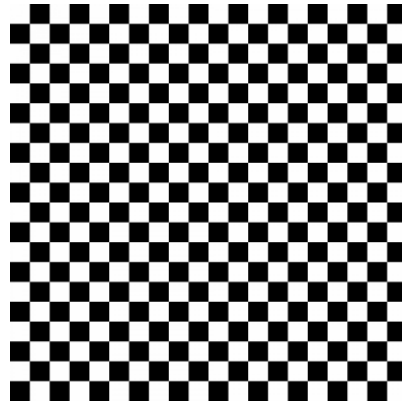
    try:
        ...
```

# [Code] 6-cube-multiple-textures

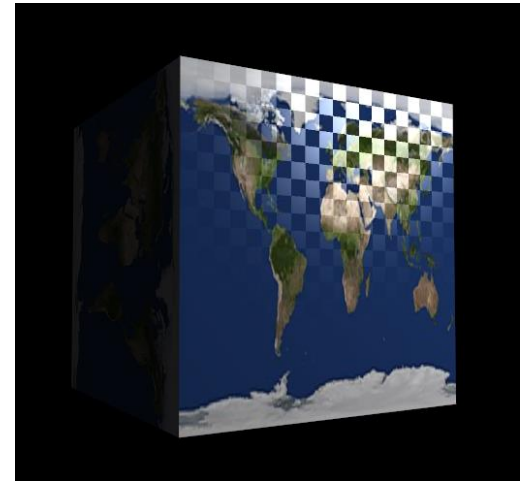
- This example loads two different images and uses them as the diffuse and specular maps respectively.



diffuse map



specular map



result

- Each texture is bound to a different texture unit, allowing both textures to be accessed in the same shader.

# Using Multiple Texture Units

---

- To use multiple textures simultaneously:
  - 1. Assign a different texture unit to each sampler.
    - `texture_diffuse -> unit 0`
    - `texture_specular -> unit 1`
  - 2. Activate each texture unit.
    - `glActiveTexture(GL_TEXTURE0)`
    - `glActiveTexture(GL_TEXTURE1)`
  - 3. Bind a texture object to each unit.
    - `glBindTexture(...)`

# [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`  
Each sampler is connected to a different texture unit.
- 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():
    ...
    glUseProgram(shader_program) # updating uniform requires activating shader prog.

    # diffuse texture
    texture_diffuse = glGenTextures(1)
    glActiveTexture(GL_TEXTURE0)
    glBindTexture(GL_TEXTURE_2D, texture_diffuse)
    ...
    try:
        img = Image.open('./320px-Solarsystemscope_texture_8k_earth_daymap.jpg')
        ...
    glUniform1i(glGetUniformLocation(shader_program, 'texture_diffuse'), 0)

    # specular texture
    texture_specular = glGenTextures(1)
    glActiveTexture(GL_TEXTURE1)
    glBindTexture(GL_TEXTURE_2D, texture_specular)
    ...
    try:
        img = Image.open('./plain-checkerboard.jpg')
        ...
    glUniform1i(glGetUniformLocation(shader_program, 'texture_specular'), 1)
```

# [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 Lab Check

---

- Let's start today's lab check.
- TAs will guide you.
- Next week will be our final lecture of the semester.
  - There will be no lab session next week.