3D Graphics Programming

T163 - Game Programming

GEORGE BROWN COLLEGE

Fall 2020

Week 9

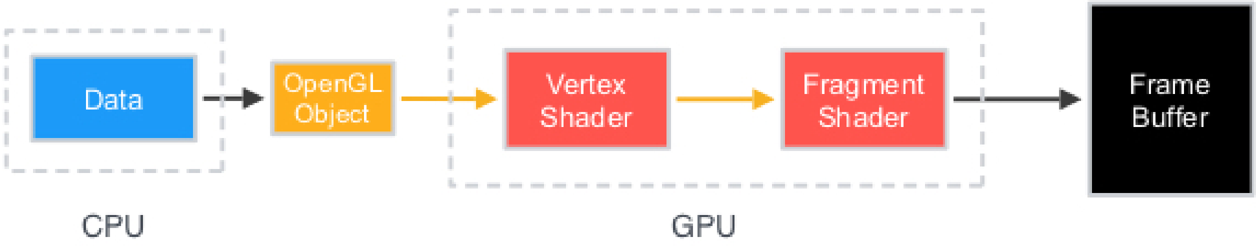
Object and App Review

Texturing

Fall 2020

Object Review

- First, let's do a brief review of objects.



Object Review

- Type of Objects:

Regular Objects

Container Objects

Object Review

- Regular Objects:

Buffer Objects

Renderbuffer Objects

Query Objects

Texture Objects

Sampler Objects

Object Review

- Container Objects:

FrameBuffer Objects

Vertex Array Objects

Transform Feedback Objects

Program Pipeline Objects

Object Review

- Regular Objects can be bound to different binding points:

GL_ARRAY_BUFFER

GL_TEXTURE_BUFFER

GL_ELEMENT_ARRAY_BUFFER

Etc…

Object Review

- The binding point decides how the object behaves
  - For buffer objects, for example:

GL_ARRAY_BUFFER -> for Vertex Buffer Objects (VBOs)

GL_TEXTURE_BUFFER -> for Texture Buffer Objects (TBOs)

GL_ELEMENT_ARRAY_BUFFER -> for Index Buffer Objects (IBOs)

Etc…

Object Review

- Why a VBO?

OpenGL can't read vertex data in CPU memory.

Vertex data must be sent to the GPU before it can be used in the rendering pipeline.

A VBO is able to take the data stored in the CPU and transmit it to the GPU.

Object Review

❖ Why a VAO?

Let's say that you need to render 12 different characters each with their own data type, offset, etc. All of this information must be prepared before it is rendered.
This is a lot of states to set and a lot of error checking that the driver will have to do.

This is where a VAO can help organize all of this information.
A VAO is a container that stores all the states needed for rendering.
It stores the information of vertex-attribute as well as the buffer object.

Object Review

- So far to describe a single renderable entity we used:

Multiple VBOs

Single IBO

- Contained in a single VAO

Object Review

- So far to describe a single renderable entity we used

- Multiple VBOs for:

Vertex Position

Vertex Color

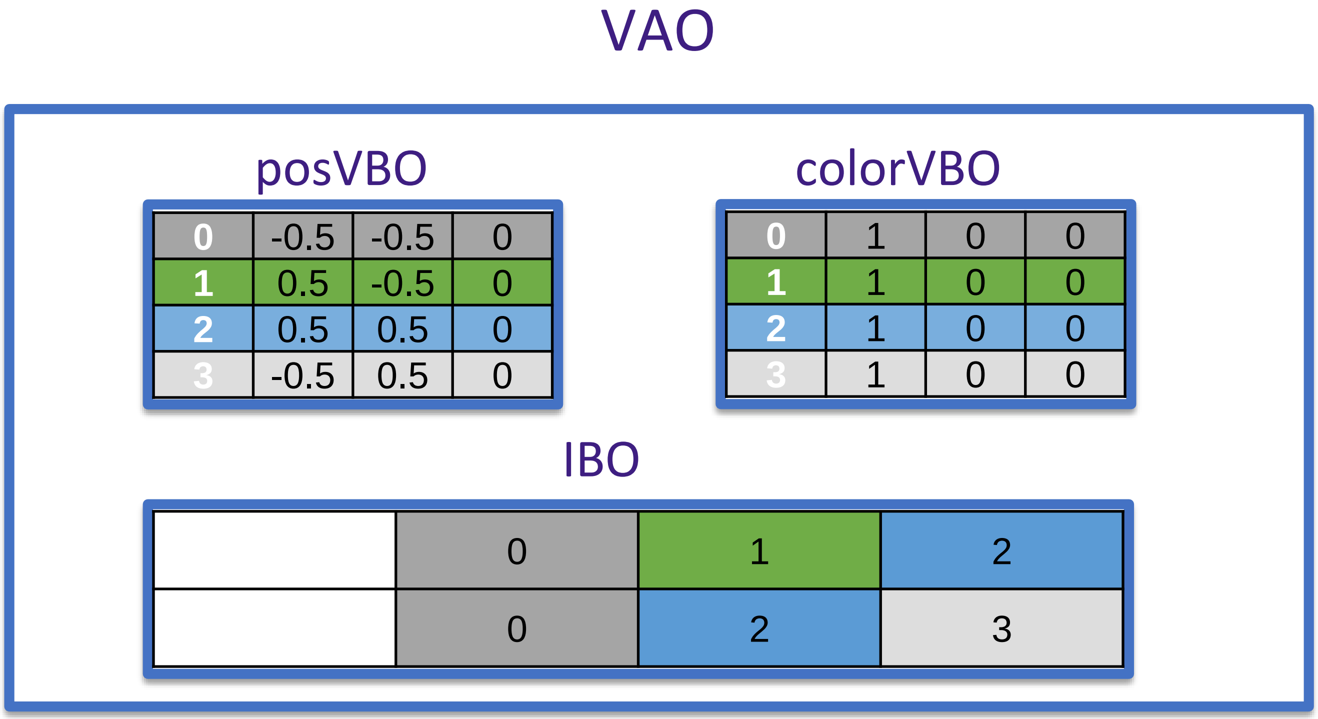
Object Review

- So far to describe a single renderable entity we used

- Single IBO for:

Index List

Object Review



VAO

posVBO

0	-0.5	-0.5	0
1	0.5	-0.5	0
2	0.5	0.5	0
3	-0.5	0.5	0

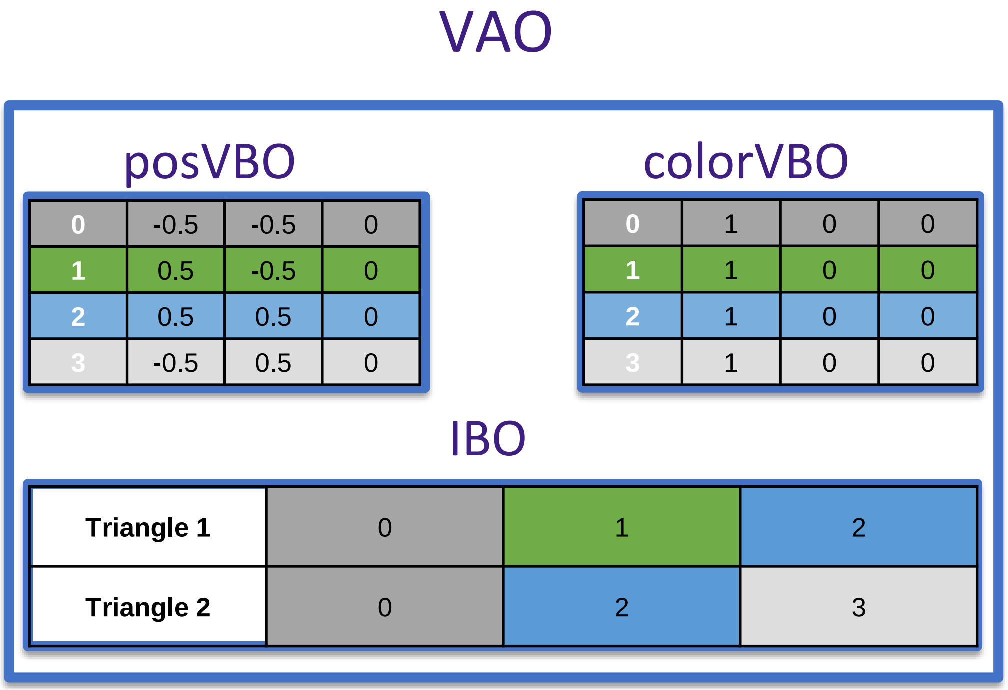
colorVBO

0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0

IBO

	0	1	2
	0	2	3

Object Review



VAO

posVBO

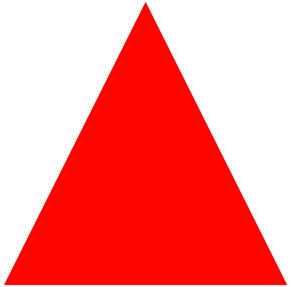
0	-0.5	-0.5	0
1	0.5	-0.5	0
2	0.5	0.5	0
3	-0.5	0.5	0

colorVBO

0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0

IBO

Triangle 1	0	1	2
Triangle 2	0	2	3



OpenGL App. Main Components

1- Main program

2- Vertex Shader

3- Fragment Shader

Texturing

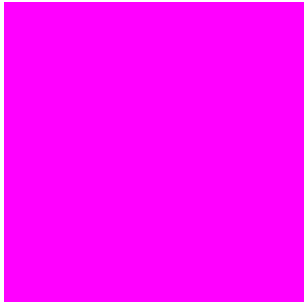
What is a Texture?

What is a Texture?

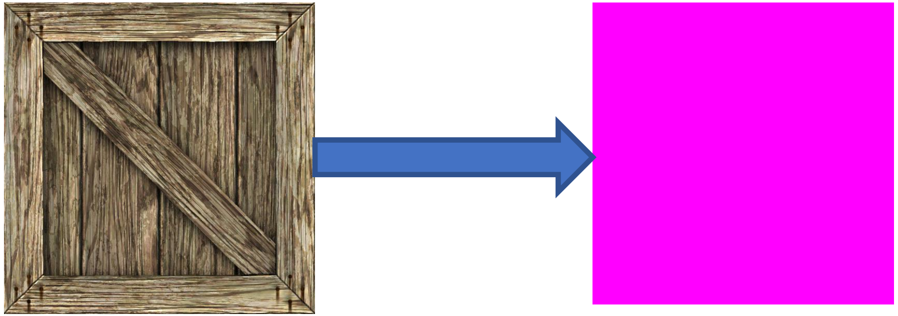
- Textures are just bytes of data.
- Each byte represents the color of a texel.

What is a Texture?

If we have a plain, textureless square:



And we want to apply this texture to it:

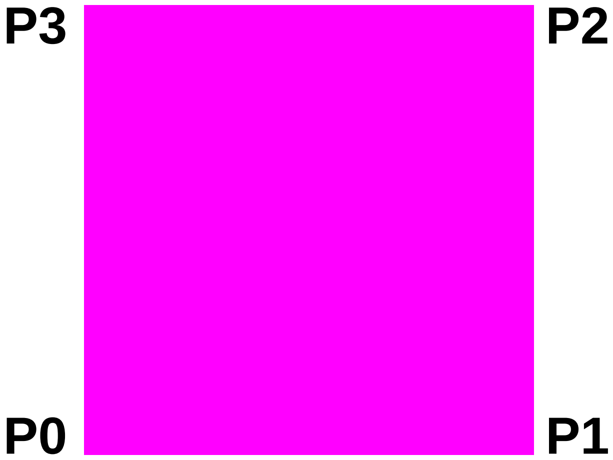


We need to know how to map them together.

What is a Texture?

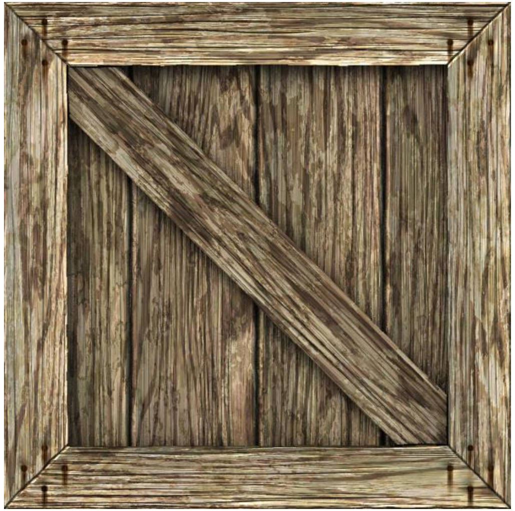
The square is made of 4 vertices.

Each vertex should map to a point on the texture.



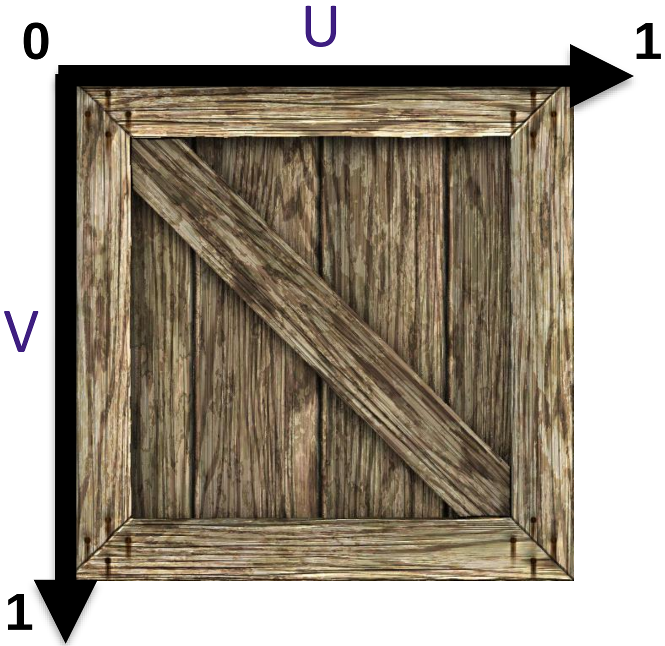
What is a Texture?

This means that we need a way to divide the texture.



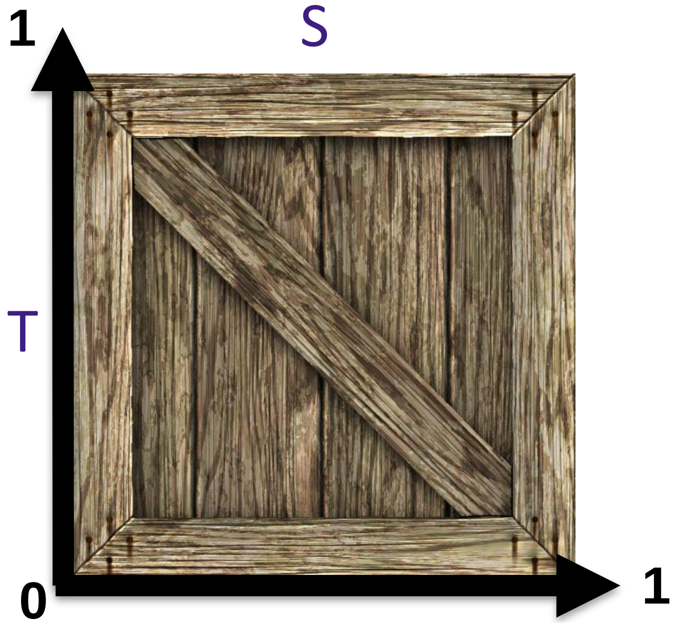
What is a Texture?

UV Mapping!!!



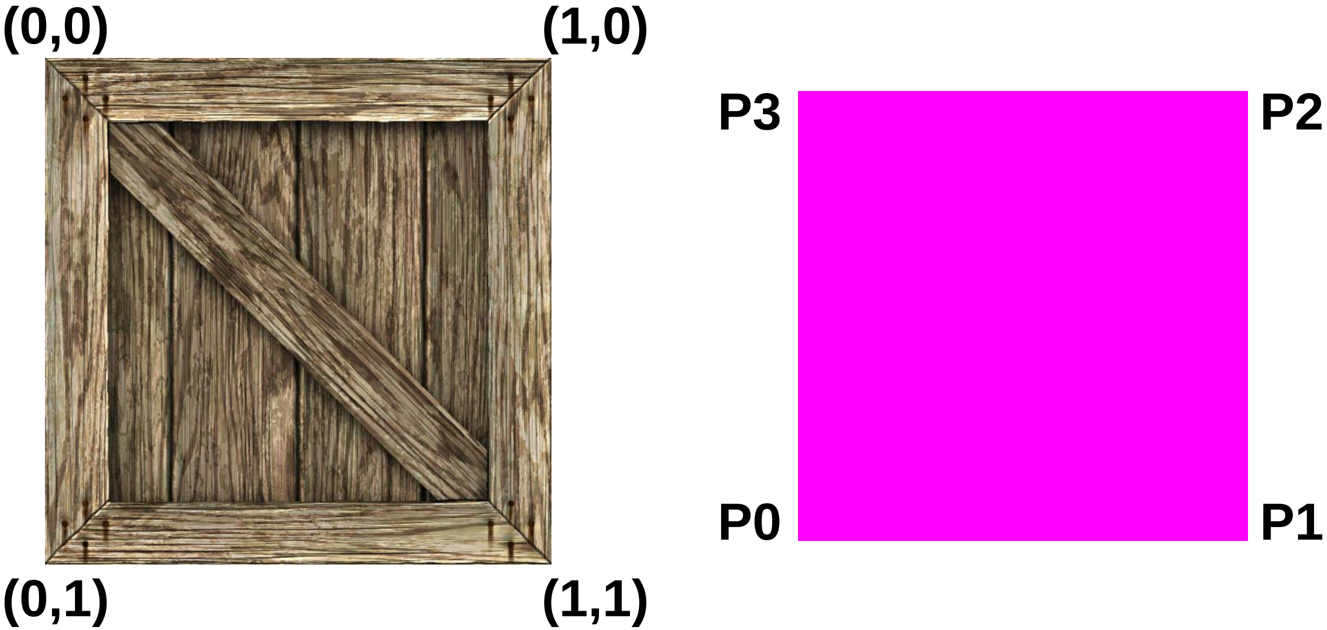
What is a Texture?

OpenGL's system is actually uses "ST" from an old Pixar standard (and OpenGL is right-handed)



What is a Texture?

Therefore, each vertex in the square gets a UV coordinate.

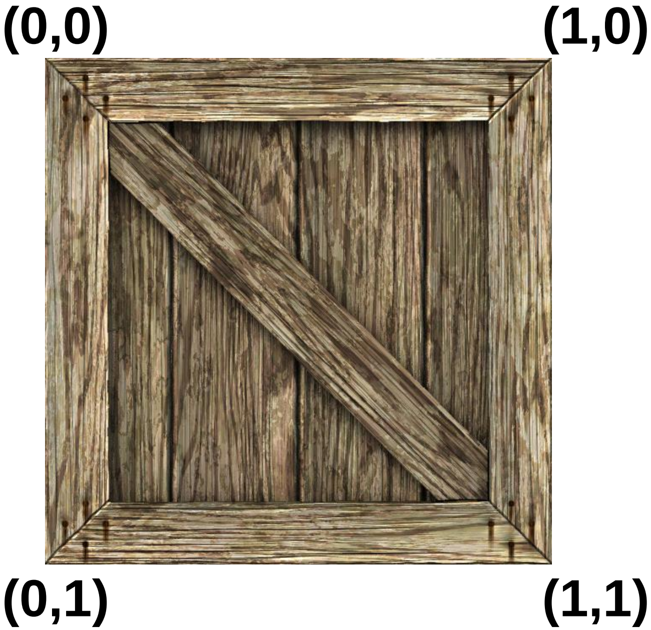


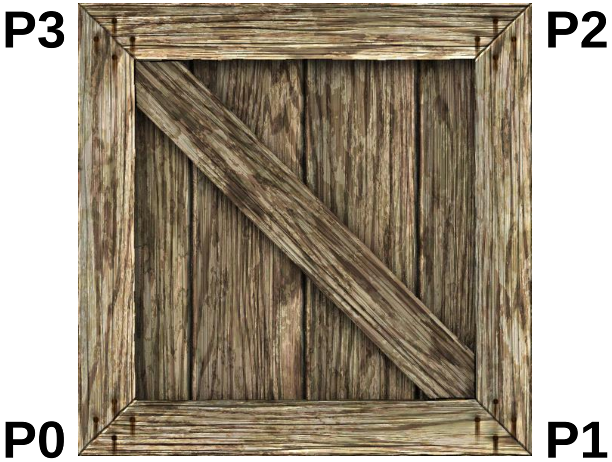
What is a Texture?

Clarify texture flipping...

What is a Texture?

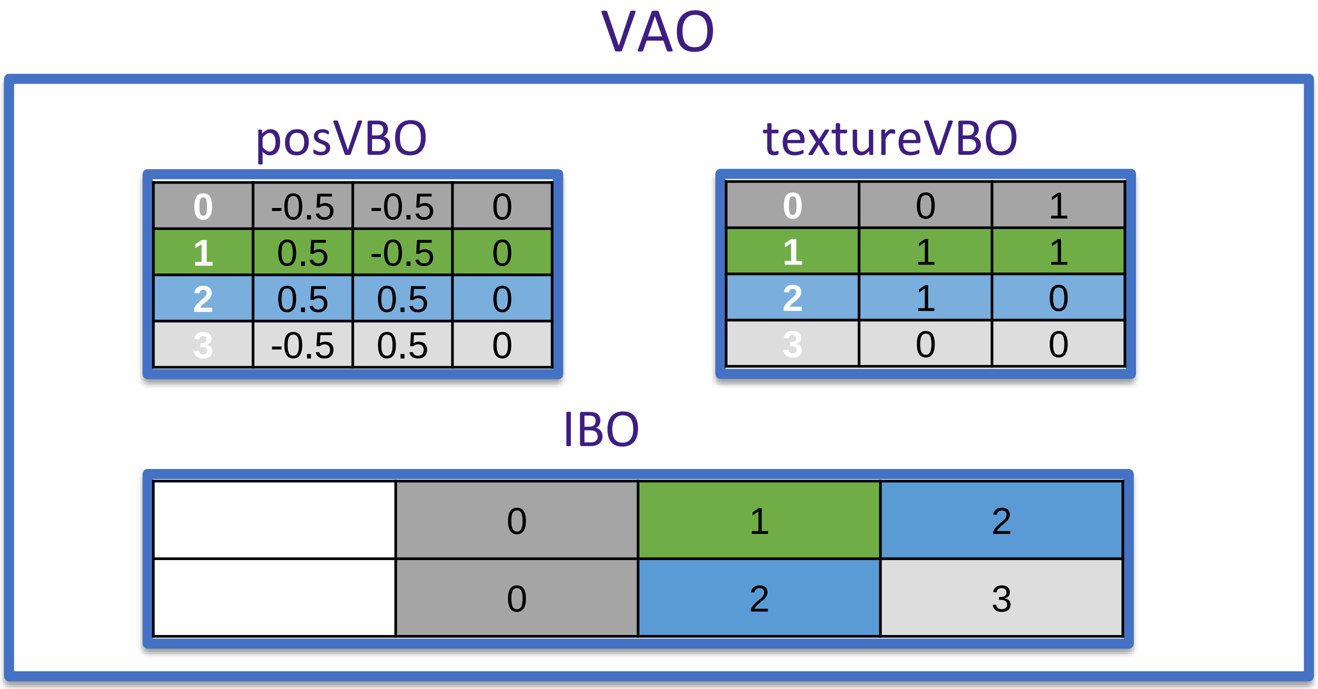
Therefore, each vertex in the square gets a UV coordinate.





What is a Texture?

Sounds easy right? So how does it work in code?



VAO

posVBO

0	-0.5	-0.5	0
1	0.5	-0.5	0
2	0.5	0.5	0
3	-0.5	0.5	0

textureVBO

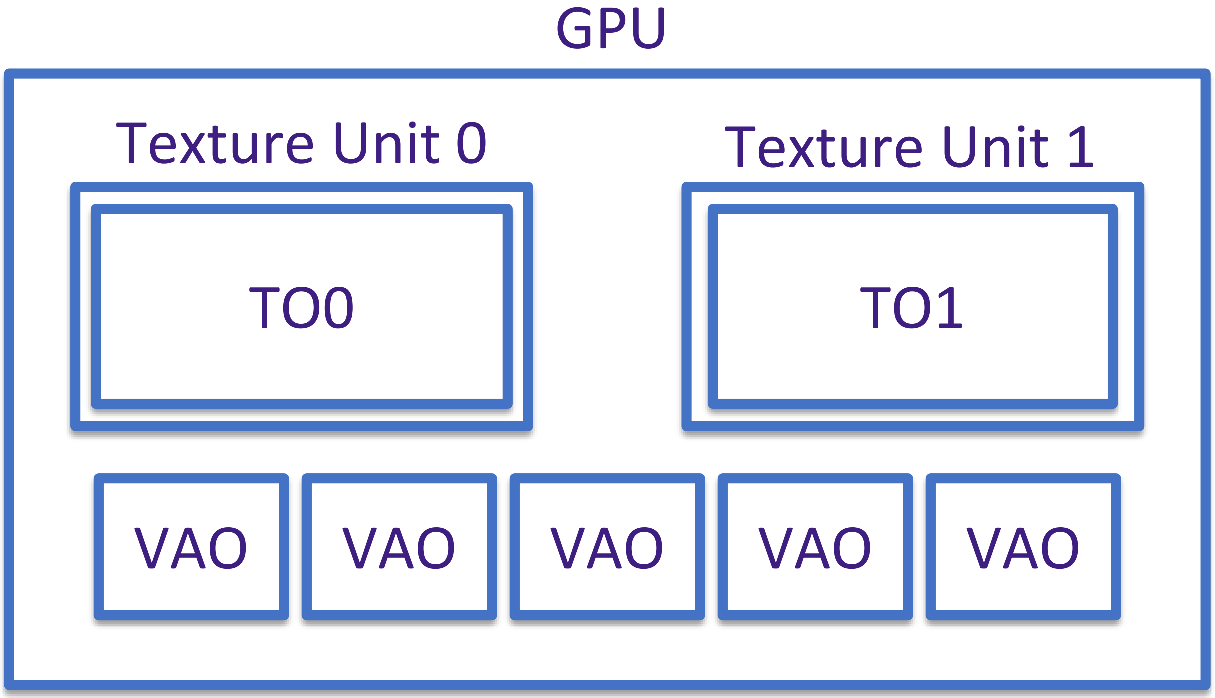
0	0	1
1	1	1
2	1	0
3	0	0

IBO

	0	1	2
	0	2	3

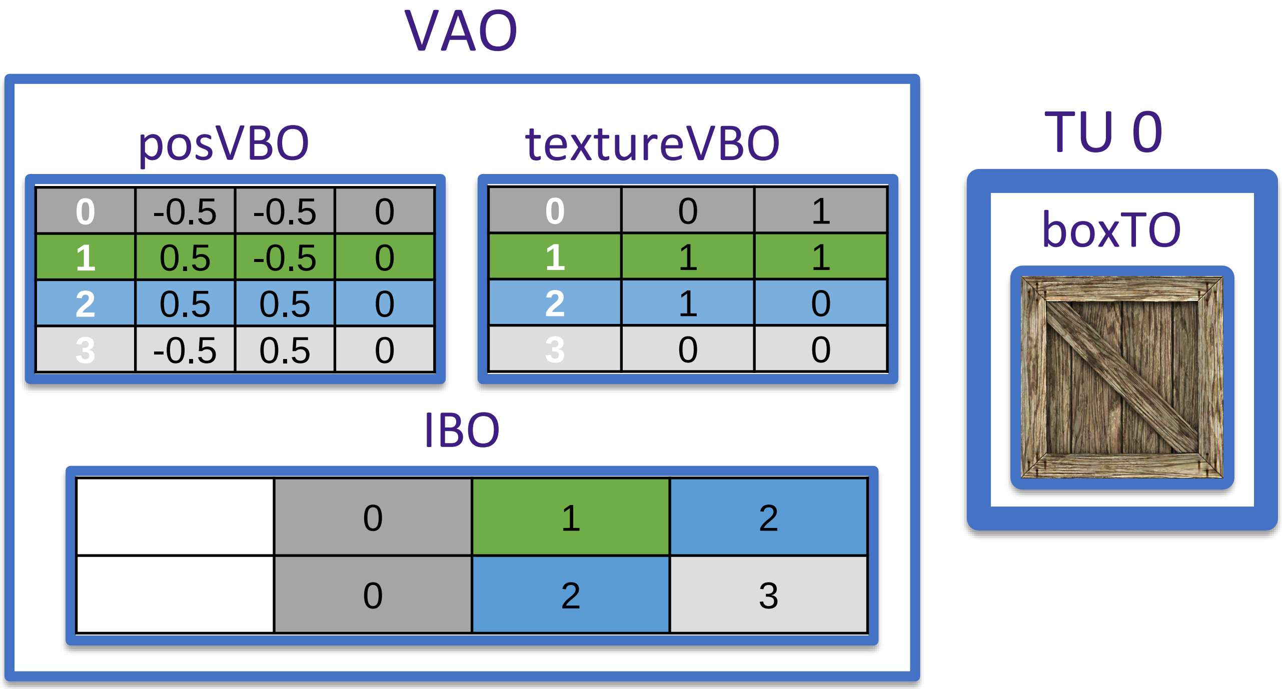
What is a Texture?

Sounds easy right? So how does it work in code?



What is a Texture?

What about the actual image?



VAO

posVBO

0	-0.5	-0.5	0
1	0.5	-0.5	0
2	0.5	0.5	0
3	-0.5	0.5	0

textureVBO

0	0	1
1	1	1
2	1	0
3	0	0

IBO

	0	1	2
	0	2	3

TU 0

boxTO

Applying Textures

Simple OpenGL Image Library

A small and easy-to-use library that loads image files directly into texture objects or creates them for you.

Applying Textures

Textures are applied in the Fragment Shader.

```
#version 410 core

in vec3 myColor;
in vec2 texCoord;
out vec4 frag_colour;

uniform sampler2D texture0;

void main() {
    frag_colour = texture(texture0, texCoord);
}
```

Applying Textures

This means that the Vertex Shader needs to change.

```
#version 410 core
layout(location = 0) in vec3 vertex_position;
layout(location = 1) in vec3 vertex_colour;
layout(location = 2) in vec2 vertex_texture;

out vec3 myColor;
out vec2 texCoord;

uniform highp mat4 MVP;

void main()
{
   myColor = vertex_colour;
   texCoord = vertex_texture;
         gl_Position = MVP * vec4(vertex_position,1.0f);
}
```

Applying Textures

How do we pass the Texture to the GPU?

```
GLint width, height;
unsigned char* image = SOIL_load_image(“/PathToImage/ImageName.png”, &width, &height, 0,
SOIL_LOAD_RGB);
```

Applying Textures

How do we pass the Texture to the GPU?

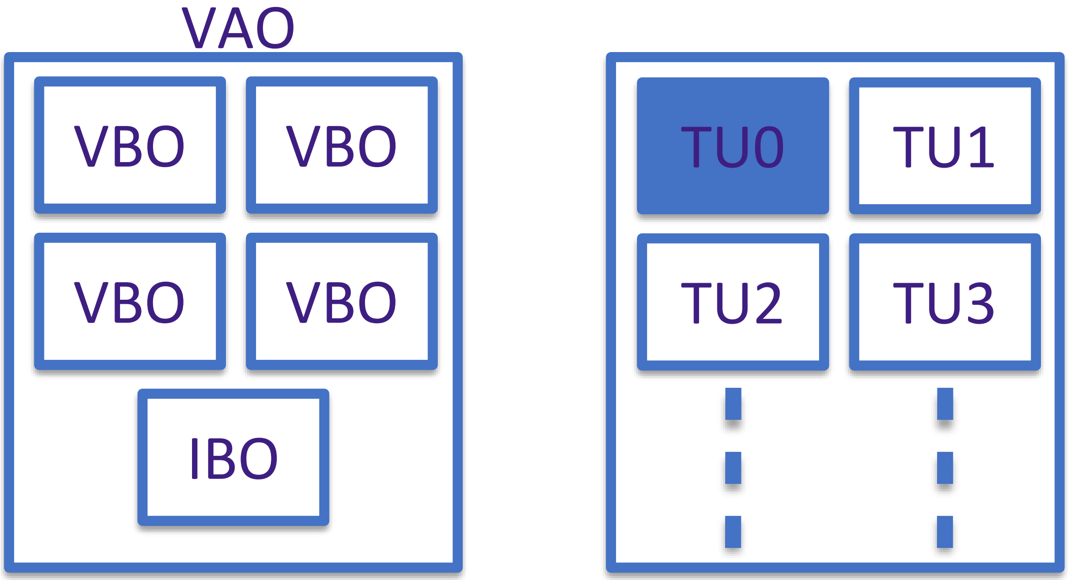
```
GLuint cube_tex = 0;
glGenTextures(1, &cube_tex);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, cube_tex);
glTexImage2D(GL_TEXTURE_2D, 0,GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, image);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glUniform1i(glGetUniformLocation(program, "texture0"), 0);
```

Applying Textures

How do we pass the Texture to the GPU?

```
glActiveTexture(GL_TEXTURE0);
```

Activates Texture Unit 0

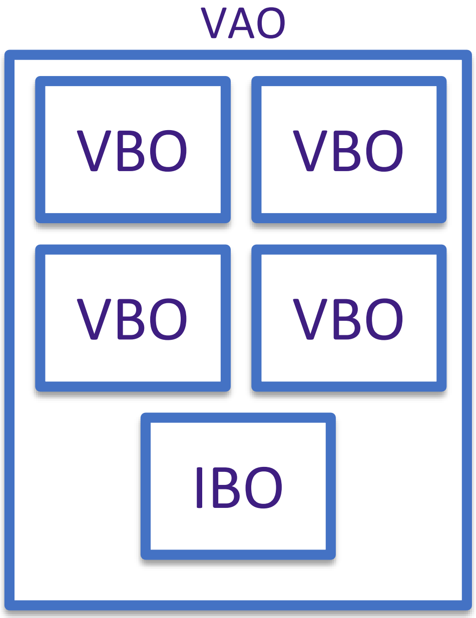


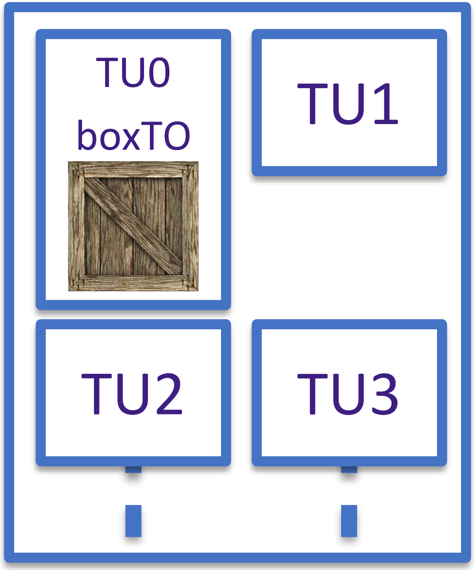
Applying Textures

How do we pass the Texture to the GPU?

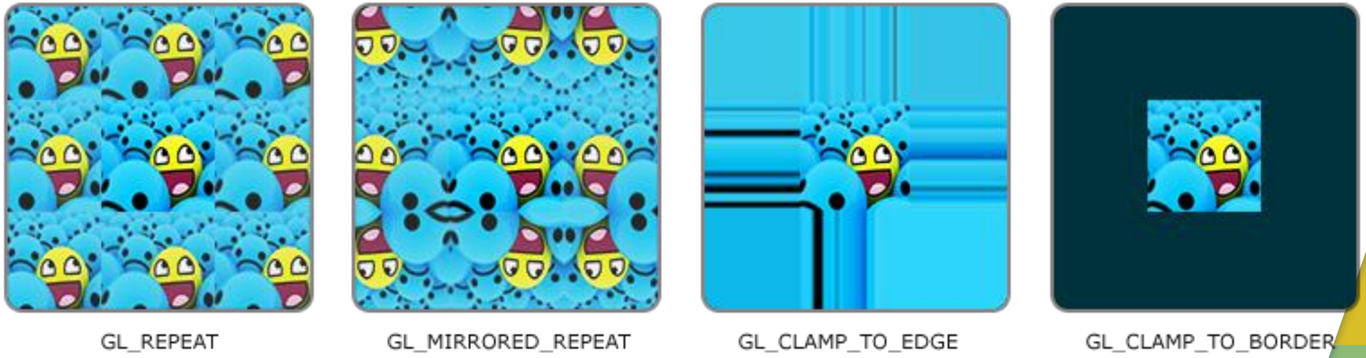
```
glBindTexture(GL_TEXTURE_2D, cube_tex);
glTexImage2D(GL_TEXTURE_2D, 0,GL_RGB, width, height, 0, GL_RGB,
GL_UNSIGNED_BYTE, image);
```

Binds and transfers Texture data to boxTO in Unit 0





Wrapping


GL_REPEAT
GL_MIRRORED_REPEAT
GL_CLAMP_TO_EDGE
GL_CLAMP_TO_BORDER

Wrapping

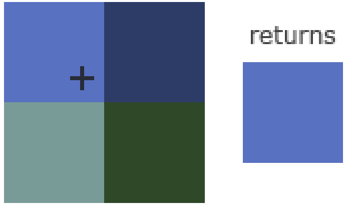
GL_REPEAT: The integer part of the coordinate will be ignored and a repeating pattern is formed

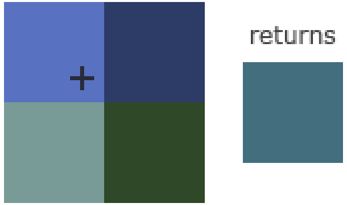
GL_MIRRORED_REPEAT: The texture will also be repeated, but it will be mirrored when the integer part of the coordinate is odd

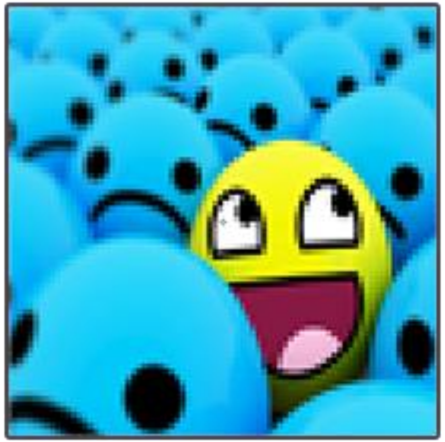
GL_CLAMP_TO_EDGE: The coordinate will simply be clamped between 0 and 1

GL_CLAMP_TO_BORDER: The coordinates that fall outside the range will be given a specified border color

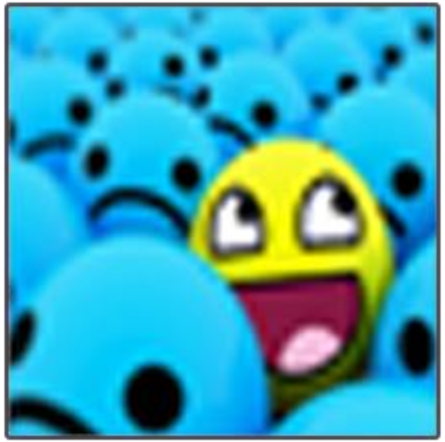
Filters?







GL_NEAREST



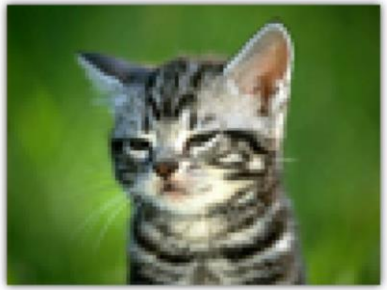
GL_LINEAR

Filters

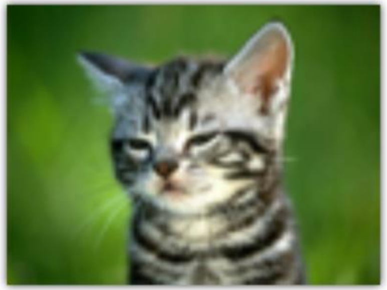
GL_NEAREST: Returns the pixel that is closest to the coordinates

GL_LINEAR: Returns the weighted average of the 4 pixels surrounding the given coordinates

GL_NEAREST_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_NEAREST and GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_LINEAR: Sample from mipmaps instead



GL_NEAREST



GL_LINEAR

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

Sets the texture properties

```
glUniform1i(glGetUniformLocation(program, "texture0"), 0);
```

Links the texture to the fragment shader

```
#version 410 core

in vec3 myColor;
in vec2 texCoord;
out vec4 frag_colour;

uniform sampler2D texture0;

void main() {
   frag_colour = texture(texture0, texCoord);
}
```

Week 9

Lab Activities

Week 9 Lab

- For the lab, see Hooman's material (with video)
- OpenGL examples covered:
    - Textures

Week 9

End