Discussion 3

PPM loading
Texture rendering in OpenGL

PPM Loading - Portable PixMap format

1. Code for **`loadPPM()`**:
   http://ivl.calit2.net/wiki/images/0/09/Loadppm.txt
2. ppm file format:
   Header:

```
1 P6
2 # Created by IrfanView
3 2048 2048
4 255
```

   1. P6: **byte format**. P3: ASCII text
   2. # comments
   3. The image **width** & **height**
   4. Maximum value of colour components for pixels
      e.g. (0...255) color values here

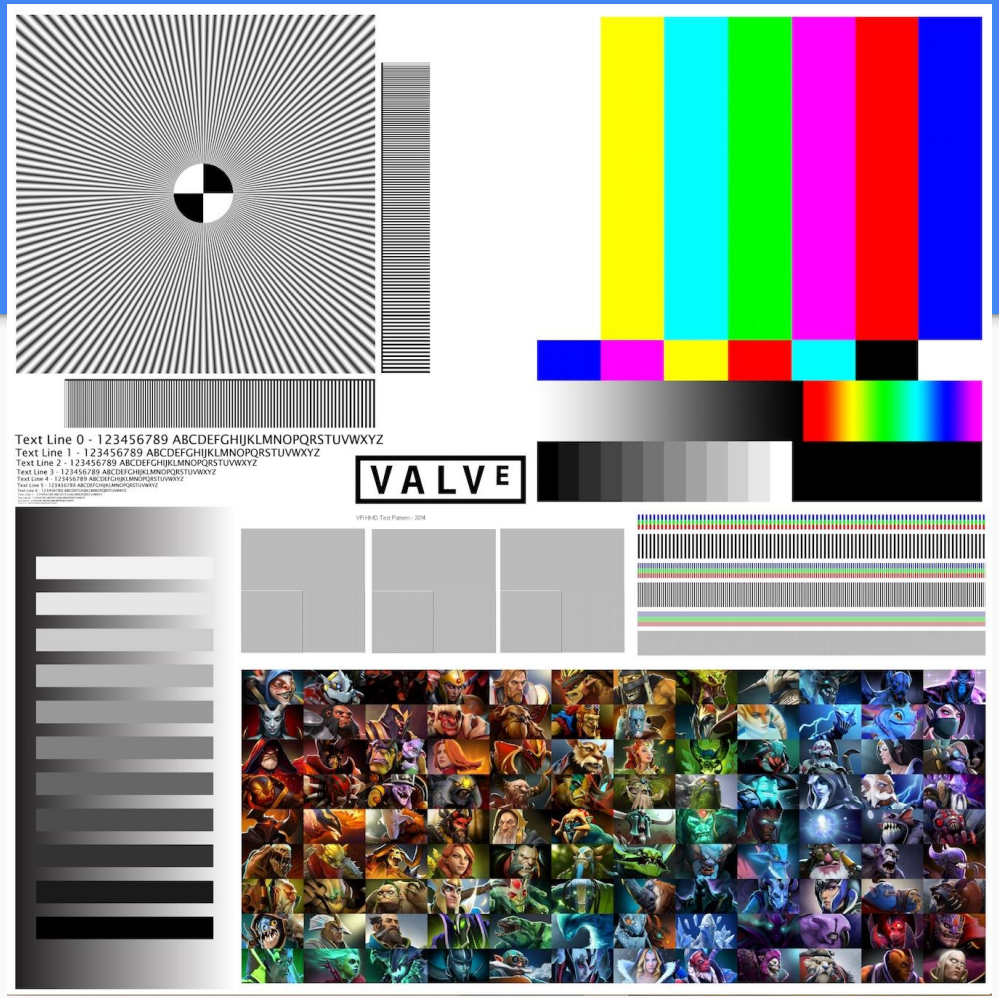
3. One byte per color in the order of **R, G, B**, 0=black, 255=white.
4. Standard convention: **top to bottom left to right order**.

PPM Loading - loadPPM() explained

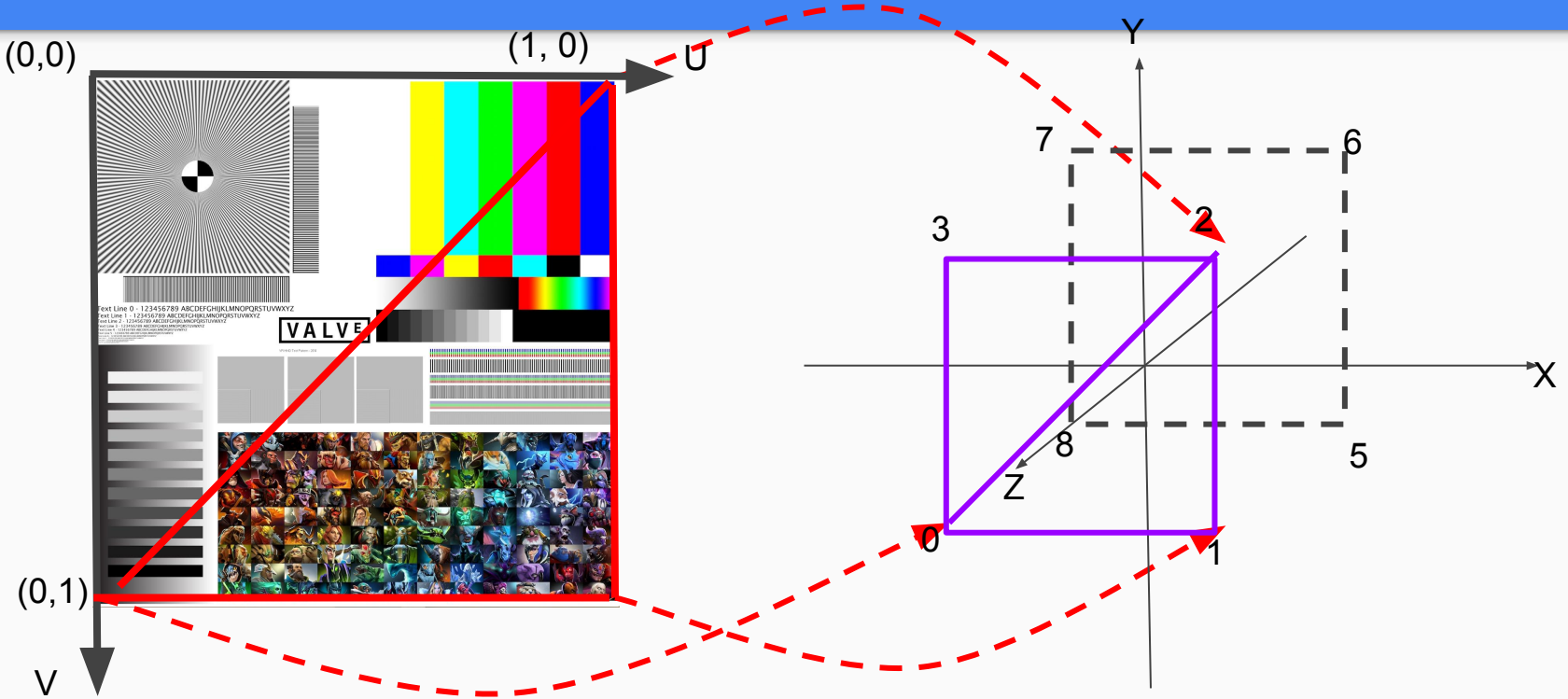
1. Use **`fgets`** to read lines and skip the comments
2. Read width and height using **`sscanf`** of format “%s %s” and **`stoi`**
3. Skip the maximum value of color
4. Use **`fread`** to read all the remaining bytes of size N = width *height*3
5. Returns **`unsigned char* rawData`**, which is a unsigned byte array of size N. (Do not forget to delete this after passing it to GL)

vr_test_pattern.ppm

Rendered image result:



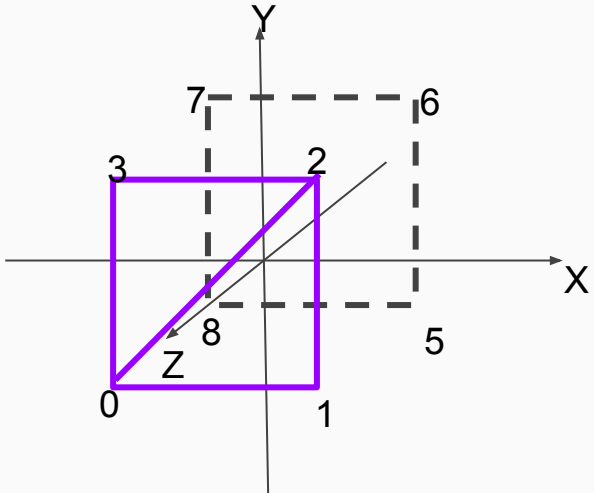
Texture Rendering - UV space

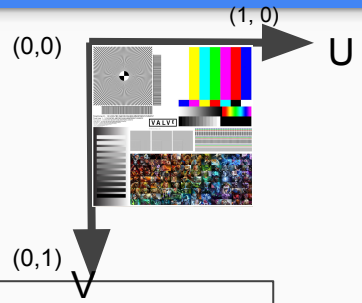


Texture Rendering - UV coordinates sample

```
const GLfloat vertices[...][3] = {
        // "Front" vertices
        { -10.0, -10.0,  10.0 },{ 10.0, -10.0,  10.0 },{ 10.0,  10.0,  10.0 },{ -10.0,  10.0,  10.0 },
        // "Back" vertices
        ....
}

const GLuint indices[6][6] = {
        // Front face
        { 0, 1, 2, 2, 3, 0},
        ...
}
```






```
const GLfloat uvs[48]= {
        // Front face
        0.0f, 1.0f, // 0
        1.0f, 1.0f, // 1
        1.0f, 0.0f, // 2
        0.0f, 0.0f, // 3
...
}
```

Texture Rendering - Criteria Explained

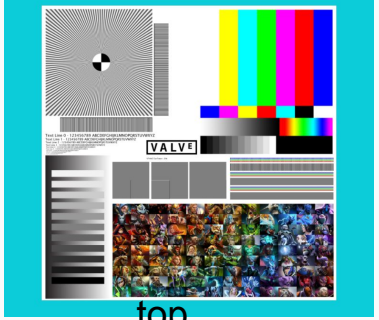
left

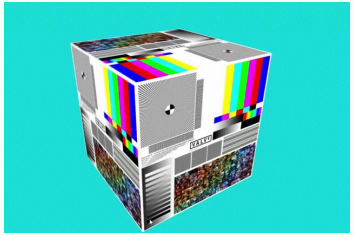
front

right

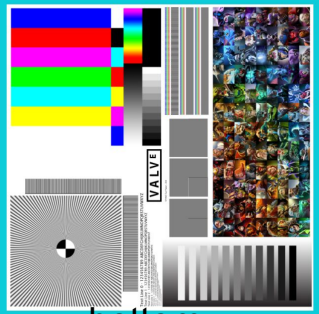
back

top





bottom

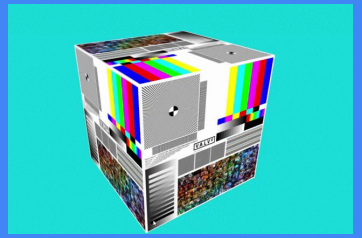


Texture Rendering - OpenGL

1. Create texture is similar with creating vertex buffers.
   After acquiring the `width`, `height` and `data`:

```
GLuint texture_ID;
glGenTextures(1, texture_ID);
glBindTexture(GL_TEXTURE_2D,  texture_ID);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, _width, _height, 0, GL_RGB,
GL_UNSIGNED_BYTE, _data);
// Setup filtering (explained on slide 13)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

Texture Rendering - OpenGL

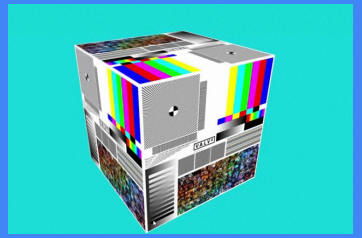


2. Setup shader - vertex

In vertex shader, add the following:

```
...
// Assuming that you used locatio=0 for position of vertices.
layout(location=1) in vec2 vertexUV;
// Output data; will be interpolated for each fragment.
out vec2 UV;
// in main function:
...
	// UV of the vertex. No special space for this one.
	UV = vertexUV;
...
```

Texture Rendering - OpenGL

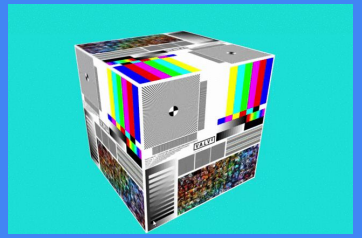


2. Setup shader - fragment

In fragment shader, add the following:

```
...
// Interpolated values from the vertex shaders
in vec2 UV;
// Output data
out vec3 color;
// fragment shader needs to know which texture to access.
uniform sampler2D myTextureSampler;
// there are 4 fragment in a pixel because we used 4x AA.
...
// In main function replace your original color assignment with the following.
        color = texture( myTextureSampler, UV).rgb;
```

Texture Rendering - OpenGL



3. Setup texture sample handle to be used for sending texture to the shader.

```
// Get a handle for our "myTextureSampler" uniform, and an ID for uv coordinates.
GLuint textureSpl_handle = glGetUniformLocation(shaderProgram, "myTextureSampler");
```

4. Setup uv handle is the same with creating vertex buffers.

```
GLuint uv_ID;
glGenBuffers(1, &uv_ID);
glBindBuffer(GL_ARRAY_BUFFER, uv_ID);
glBufferData(GL_ARRAY_BUFFER, sizeof(uvs), uvs, GL_STATIC_DRAW);
```

Texture Rendering - OpenGL

4. Use our previous setups in the render loop, function draw():

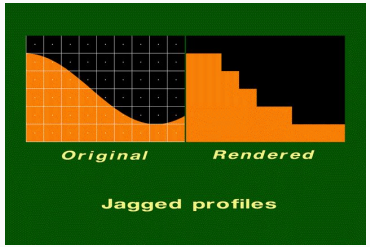
```
// Bind texture in Texture Unit 0
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture_ID);
// Set our "myTextureSampler" sampler to user Texture Unit 0
glUniform1i(textureSpl_handle, 0);
…
// Second attribute buffer: UVs. assuming you were using attribArry(0) for vertex positions.
glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER, uv_ID);
....
// Do not forget to disable this one the same way you disable AttribArray(0)
glDisableVertexAttribArray(0);
```

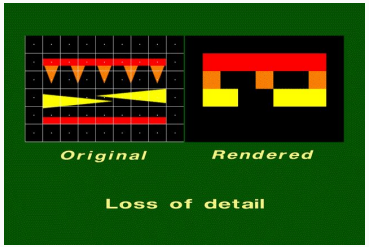
Filtering and Mipmapping

Recall when we set up the textures, we called:

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

```
GL_TEXTURE_MAG_FILTER: magnifying image.
(GL_NEAREST, GL_LINEAR)
GL_TEXTURE_MIN_FILTER: minifying image.
(GL_NEAREST, GL_LINEAR GL_NEAREST_MIPMAP_NEAREST,
 GL_LINEAR_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR,
 GL_LINEAR_MIPMAP_LINEAR)
```





GL_NEAREST: uses the value of the texture element that is nearest to the center of the pixel.
GL_LINEAR: looks around, and mixes the colours according to the distance to each other. Avoids the hard edges.

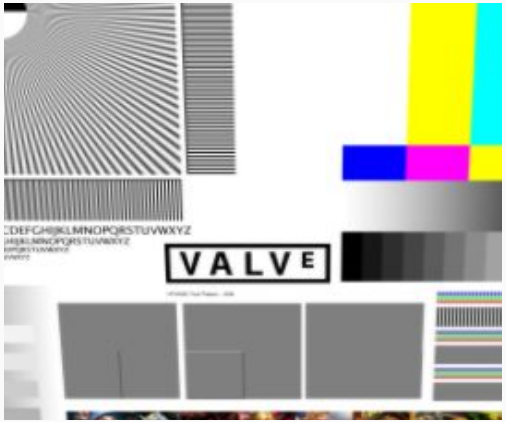
Filtering and Mipmapping

Mipmap:
are pre-calculated, optimized sequences of images, each of which is a progressively lower resolution representation of the same image. (wiki)

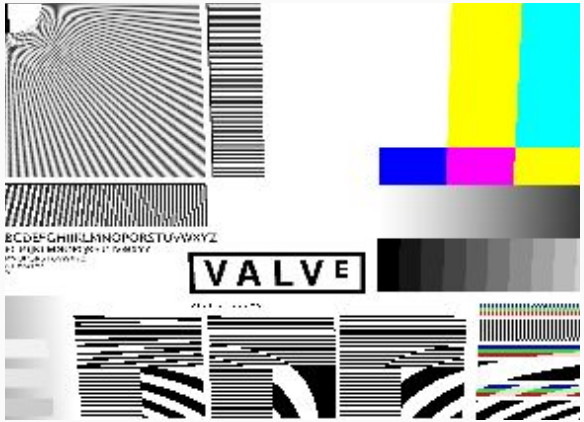
For linear filtering, “if the texture is seen from far away, mixing only 4 texels won’t be enough. Actually, if your 3D model is so far away than it takes only 1 fragment on screen, ALL the texels of the image should be averaged to produce the final color. This is obviously not done for performance reasons.”

GL_[X]_MIPMAP_[Y]: Y - chooses one(NEAREST) or two mipmaps(LINEAR), X - how to use the value.

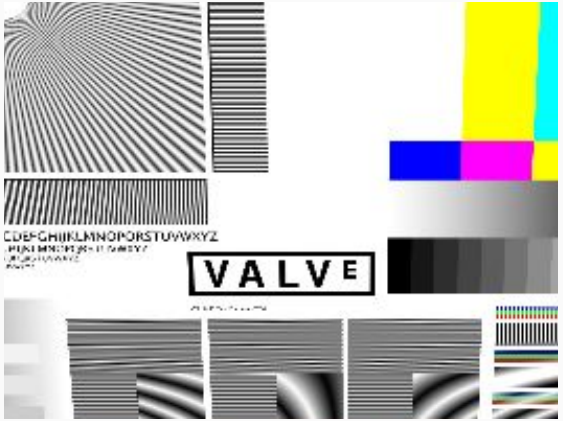
GUESS which one?
GL_NEAREST, GL_LINEAR, or GL_LINEAR_MIPMAP_LINEAR



A



B



C

Tips

1. USE linear for MAG and mipmap for MIN! And don’t forget to add one more line:
   ```
   // Generate mipmaps, by the way.
   glGenerateMipmap(GL_TEXTURE_2D);
   ```

2. Add some key_callback code if you are working without the HMD first, e.g. 4:left, 6: right, 8: top, 2: bottom, 5: back, space:front. Very useful for debugging your UV coordinates.

References

1. loadPPM(): http://ivl.calit2.net/wiki/images/0/09/Loadppm.txt
2. Texture rendering tutorial from OpenGL: http://www.opengl-tutorial.org/beginners-tutorials/tutorial-5-a-textured-cube/
3. More GL review: http://www.opengl-tutorial.org/beginners-tutorials/
4. Tutorial with complete sample code for skybox: https://learnopengl.com/code_viewer.php?code=advanced/cubemaps_reflection