Tutorial 5

TA: Mengyun Liu, Hongtu Xu

Agenda

- **Quiz 1**
- **Area light**
  - Formulation
  - Implementation in OpenGL
- **Texture**
  - Texture storage
  - Implemention in OpenGL

Quiz 1

Quiz 1

Problem 1

You are given a camera in the world space. The camera is looking at a point whose position is $\boldsymbol{g}$, while the position of the camera is $\boldsymbol{e}$. In addition, the reference up vector of the camera is $\boldsymbol{u}$. You can assume that all the vectors are unit and column vectors. Please calculate:

(1) The local coordinates of the camera (right-handed).

(2) Given a point P whose position is $\boldsymbol{p} = (x_0, y_0, z_0)^T$ in the world space. Calculate its position $\boldsymbol{p}'$ in the view space. (you can leave the answer as the multiplication of some matrices and vectors).

Quiz 1

You are given a camera in the world space. The camera is looking at a point whose position is $\boldsymbol{g}$, while the position of the camera is $\boldsymbol{e}$. In addition, the reference up vector of the camera is $\boldsymbol{u}$. You can assume that all the vectors are unit and column vectors. Please calculate:

(1) The local coordinates of the camera (right-handed).

(1) View direction $\tilde{\boldsymbol{g}} = \text{normalize}(\boldsymbol{g} - \boldsymbol{e})$
Right direction $\tilde{\boldsymbol{r}} = \text{normalize}(\tilde{\boldsymbol{g}} \times \boldsymbol{u})$
Up direction $\tilde{\boldsymbol{t}} = \tilde{\boldsymbol{r}} \times \tilde{\boldsymbol{g}}$ The coordinates are:

- $+x : \tilde{\boldsymbol{r}}$
- $+y : \tilde{\boldsymbol{t}}$
- $+z : -\tilde{\boldsymbol{g}}$

Quiz 1

You are given a camera in the world space. The camera is looking at a point whose position is $\boldsymbol{g}$, while the position of the camera is $\boldsymbol{e}$. In addition, the reference up vector of the camera is $\boldsymbol{u}$. You can assume that all the vectors are unit and column vectors. Please calculate:

(2) Given a point P whose position is $\boldsymbol{p} = (x_0, y_0, z_0)^T$ in the world space. Calculate its position $\boldsymbol{p}'$ in the view space. (you can leave the answer as the multiplication of some matrices and vectors).

(2) $\boldsymbol{M}_{\text{view}}$ can be decomposed into rotation $\boldsymbol{R}_{\text{view}}$ and translation $\boldsymbol{T}_{\text{view}}$, $\boldsymbol{M}_{\text{view}} = \boldsymbol{R}_{\text{view}}\boldsymbol{T}_{\text{view}}$.

$$\boldsymbol{T}_{\text{view}} = \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}, \boldsymbol{R}_{\text{view}} = \begin{bmatrix} \tilde{\boldsymbol{r}} & \tilde{\boldsymbol{t}} & -\tilde{\boldsymbol{g}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^T$$

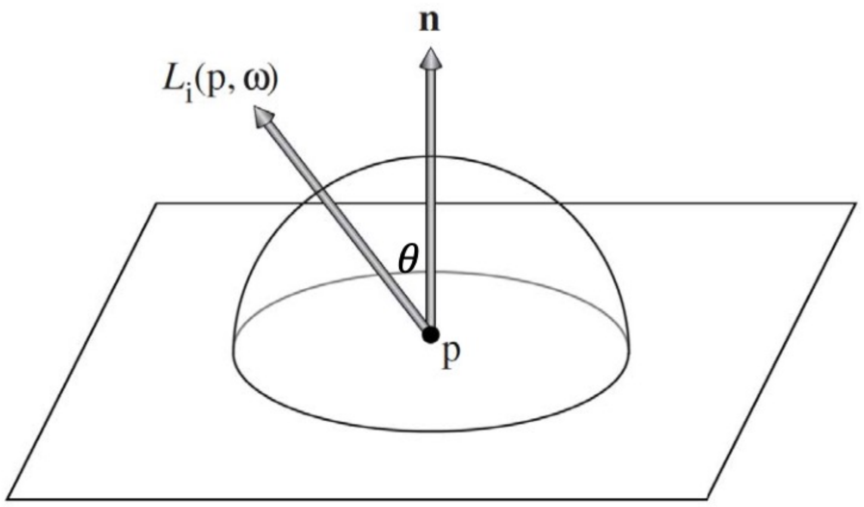
$$\begin{bmatrix} \boldsymbol{p}' \\ 1 \end{bmatrix} = \boldsymbol{M}_{\text{view}} \begin{bmatrix} \boldsymbol{p} \\ 1 \end{bmatrix}$$

Area light

Computing the irradiance of a vertex

- **Computation of irradiance *E***
  - Irradiance at a point p with surface normal **n** due to radiance over a set of directions **Ω** is

$$E(\mathrm{p}, \mathbf{n}) = \int_{\Omega} L_{\mathrm{i}}(\mathrm{p}, \omega)\, |\cos\theta|\, \mathrm{d}\omega$$

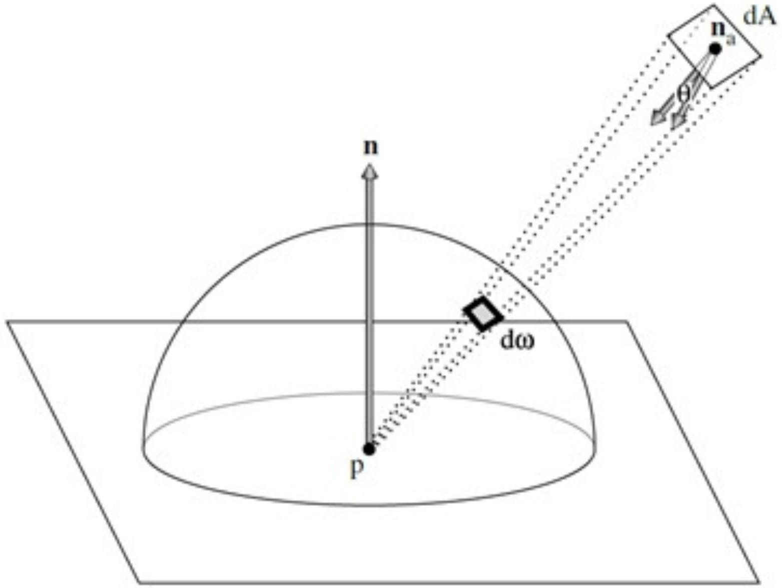


Computing the radiance of a vertex

- **Integral over area**
  - Turn integrals over directions into integrals over areas
  - Irradiance will be much easier to compute over area
  - Differential area is related to differential solid angle:

$$\mathrm{d}\omega = \frac{\mathrm{d}A \ \cos\theta}{r^2}$$

Distance to p



Implementation of area light in OpenGL

- Idea: To sum up the radiance over the area light

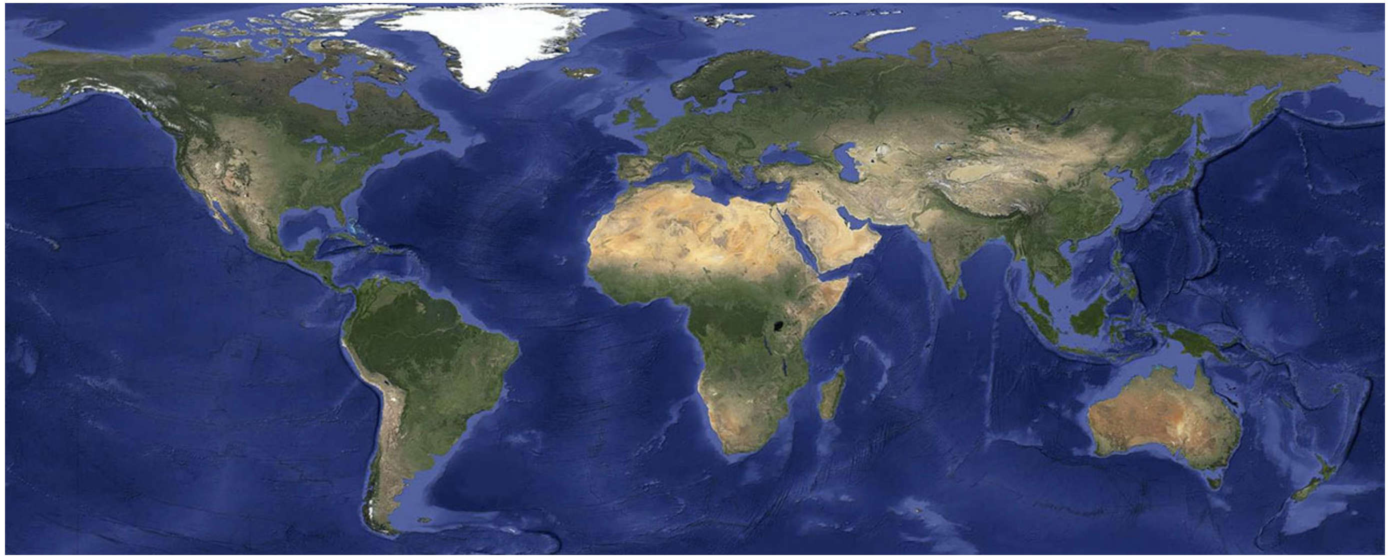
$$L_o(p, \omega_o) = \int_A f(p, \omega_o, \omega_i)\, L_i(p, \omega_i)\, |\cos\theta_i|\, \frac{dA\ \cos\theta}{r^2}$$

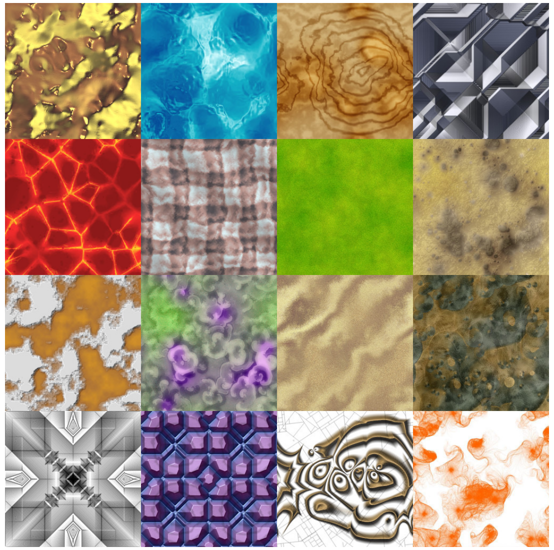
- Step:
  - Sample from area
  - For each sample, compute the irradiance
  - Add each irradiance up
- How?
  - Use a shader
  - Pass the samples to the fragment shader by utilizing the uniform variables

Texture

Texture Map

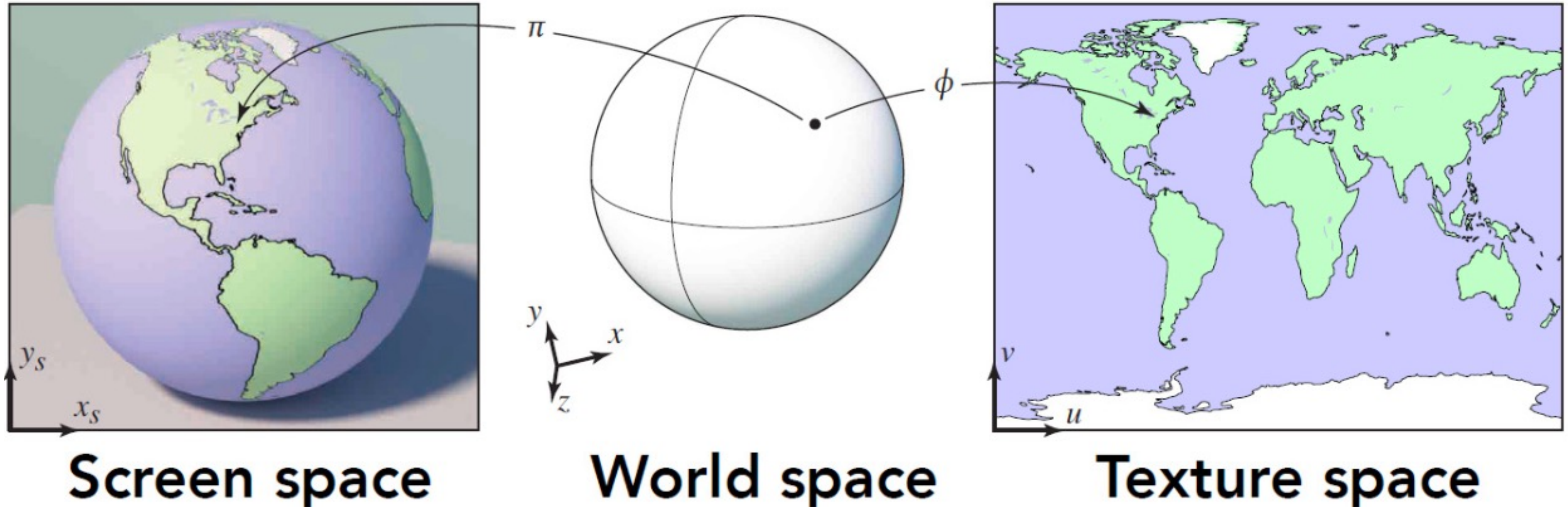
- An image applied to the surface
  - Bitmap image
    - Directly stored data
  - Procedural texture
    - Created using a mathematical description





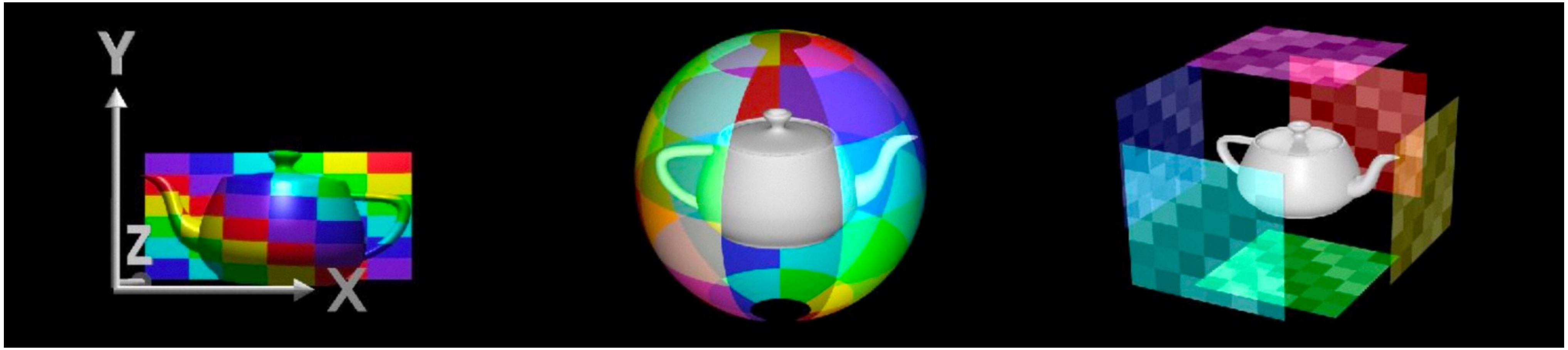
Texture Mapping

- To Map points on surface to the points on the texture
- Idea: To regard the texture color as the surface color



Texture Coordinate Functions

- To specify the mapping
- To project world space to texture space
- Many kinds of texture coordinate functions



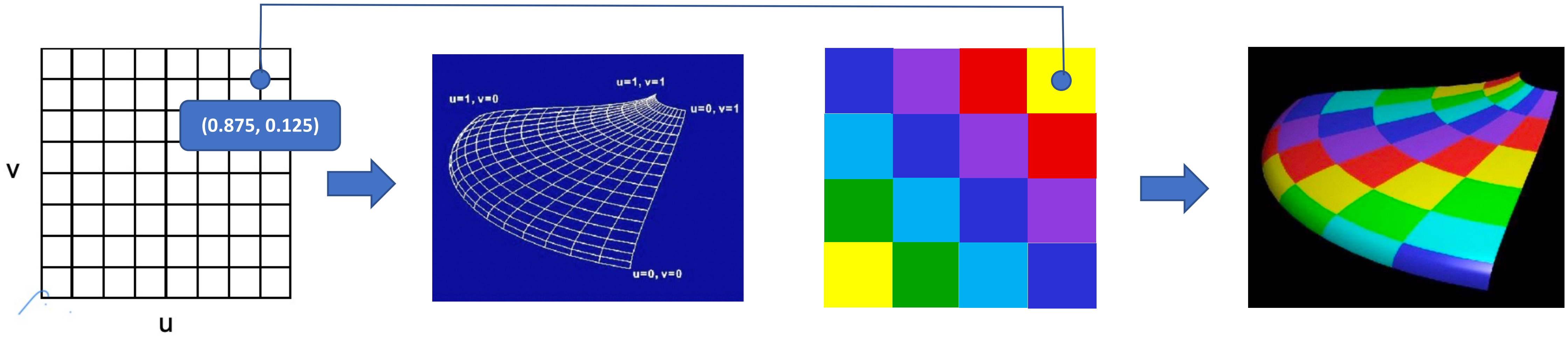
Planar projection

Spherical projection

Cube map projection

Texturing a parametric surface

- Recall:
  - We have the mapping between the parameter space and world space
- We can construct our coordinate by mapping the parameter space to the texture space.



Texture Storage

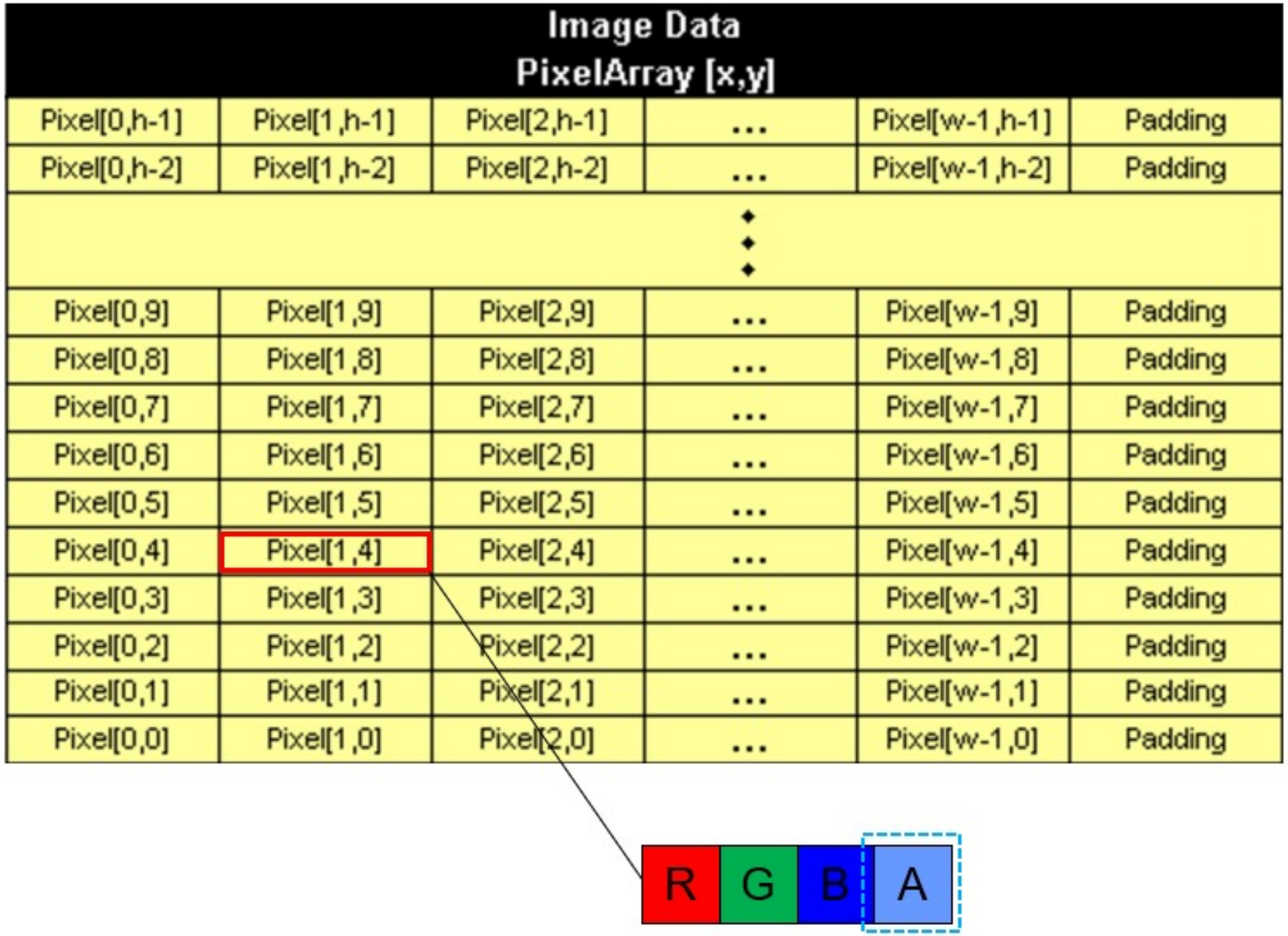


Image Data PixelArray [x,y]					
Pixel[0,h-1]	Pixel[1,h-1]	Pixel[2,h-1]	...	Pixel[w-1,h-1]	Padding
Pixel[0,h-2]	Pixel[1,h-2]	Pixel[2,h-2]	...	Pixel[w-1,h-2]	Padding
			⋮		
Pixel[0,9]	Pixel[1,9]	Pixel[2,9]	...	Pixel[w-1,9]	Padding
Pixel[0,8]	Pixel[1,8]	Pixel[2,8]	...	Pixel[w-1,8]	Padding
Pixel[0,7]	Pixel[1,7]	Pixel[2,7]	...	Pixel[w-1,7]	Padding
Pixel[0,6]	Pixel[1,6]	Pixel[2,6]	...	Pixel[w-1,6]	Padding
Pixel[0,5]	Pixel[1,5]	Pixel[2,5]	...	Pixel[w-1,5]	Padding
Pixel[0,4]	Pixel[1,4]	Pixel[2,4]	...	Pixel[w-1,4]	Padding
Pixel[0,3]	Pixel[1,3]	Pixel[2,3]	...	Pixel[w-1,3]	Padding
Pixel[0,2]	Pixel[1,2]	Pixel[2,2]	...	Pixel[w-1,2]	Padding
Pixel[0,1]	Pixel[1,1]	Pixel[2,1]	...	Pixel[w-1,1]	Padding
Pixel[0,0]	Pixel[1,0]	Pixel[2,0]	...	Pixel[w-1,0]	Padding

Implementation in OpenGL

- Creating a texture object and specifying a texture for that object

```
unsigned int texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);

// You may want to set the texture wrapping parameters
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
// You may want to set texture filtering parameters
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

int width, height, nrChannels;
unsigned char *data = YOUR_TEXTURE_IMAGE // Specify your image here
if (data) {
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
  // You may want to use mipmap
  glGenerateMipmap(GL_TEXTURE_2D);
}
```

Implementation in OpenGL

- Specifying the mapping with a shader
  - For example, we have the following data

```
float vertices[] = {
    // positions          // colors           // texture coords
     0.5f,  0.5f, 0.0f,   1.0f, 0.0f, 0.0f,   1.0f, 1.0f,   // top right
     0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f,   1.0f, 0.0f,   // bottom right
    -0.5f, -0.5f, 0.0f,   0.0f, 0.0f, 1.0f,   0.0f, 0.0f,   // bottom left
    -0.5f,  0.5f, 0.0f,   1.0f, 1.0f, 0.0f,   0.0f, 1.0f    // top left
};
```

  - And you have set the attributes correctly

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0); // position attribute
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1); // color attribute
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 * sizeof(float)));
glEnableVertexAttribArray(2); // texture coord attribute
```

Implementation in OpenGL

- Applying the texture to compute color of each vertex with a shader

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;
layout (location = 2) in vec2 aTexCoord;

out vec3 ourColor;
out vec2 TexCoord;

void main()
{
    gl_Position = vec4(aPos, 1.0);
    ourColor = aColor;
    TexCoord = aTexCoord;
}
```

Vertex shader

```
#version 330 core
out vec4 FragColor;

in vec3 ourColor;
in vec2 TexCoord;

uniform sampler2D ourTexture;

void main()
{
    FragColor = texture(ourTexture, TexCoord);
}
```

Fragment shader

Implementation in OpenGL

- If you do not want to use the shader, you can also use the built-in functions to specify the attributes
  - glVertexPointer
  - glNormalPointer
  - glColorPointer
  - glTexCoordPointer
- However, these functions may not be supported by some OpenGL versions.
  - From the OpenGL 3.3 specification, section E.2.2

Implementation in OpenGL

- Binding the texture with its id and draw your elements
- Bind before you call the drawing function

```
glBindTexture(GL_TEXTURE_2D, texture);
glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
```

Thanks