ARLAB

ME/CprE/ComS 557

Computer Graphics and Geometric Modeling

Textures

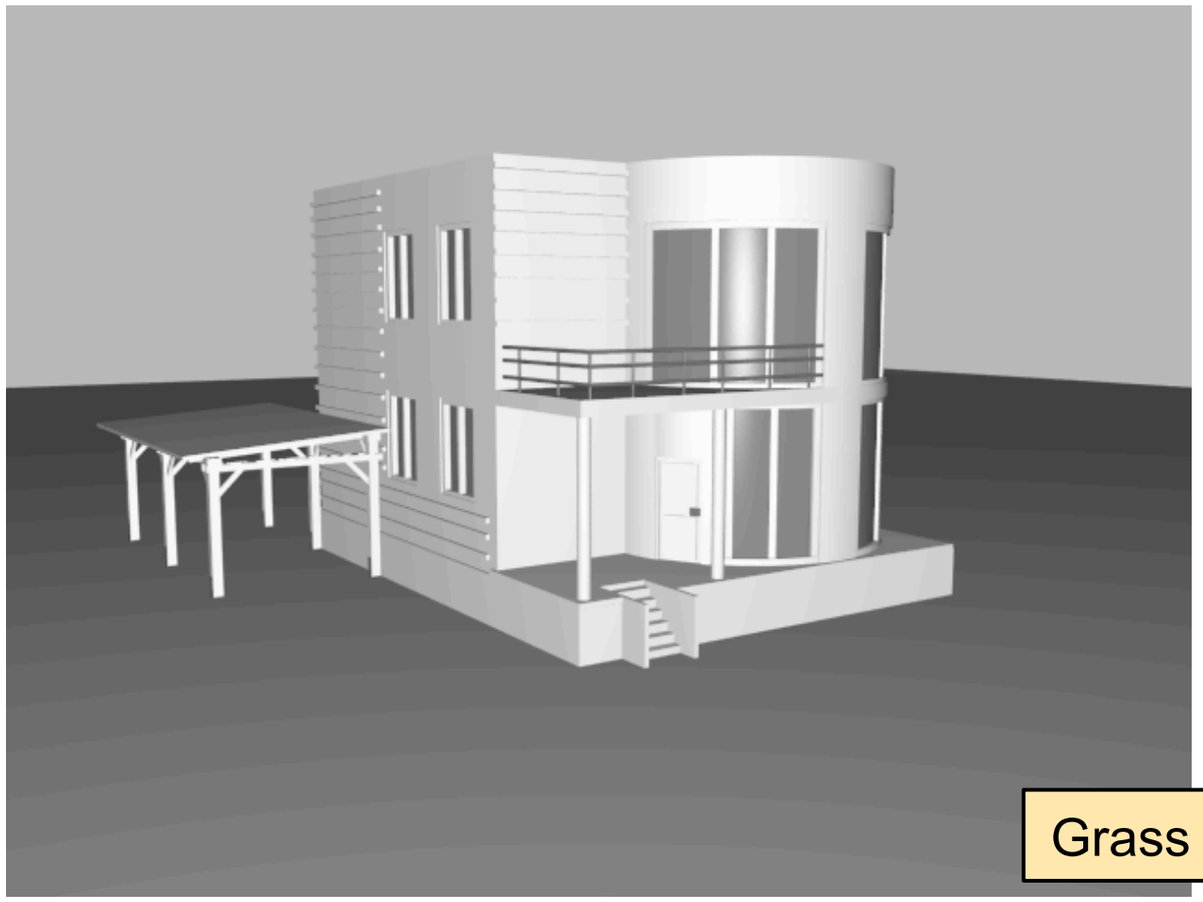
October 6th, 2015

Rafael Radkowski

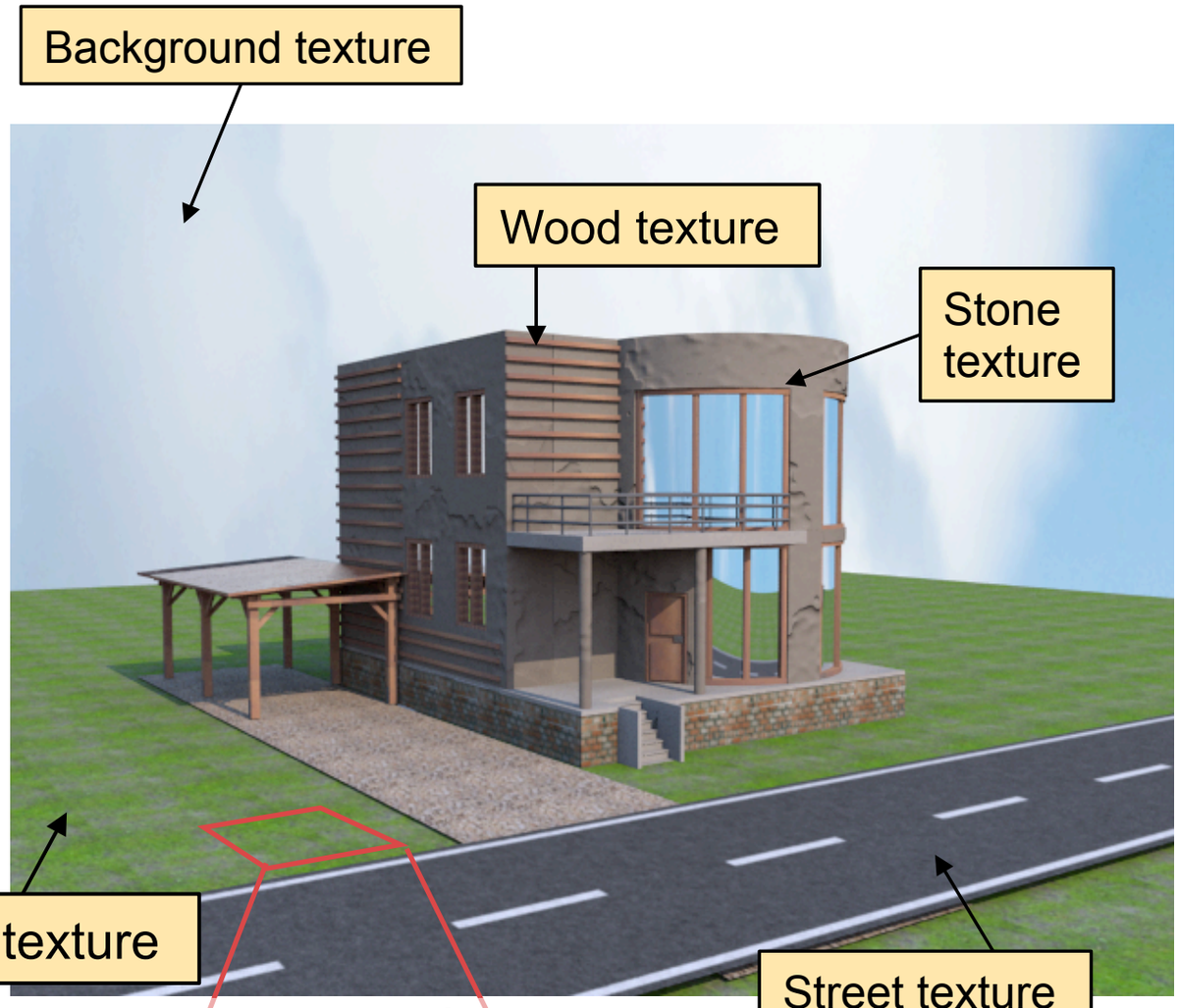
IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

What is a texture?

VRAC | HCI



Plain 3D model of a building



Textured model of a building

What is a texture?

Textures are rectangular arrays of color data, which are called **Texels (Texture Element)**.

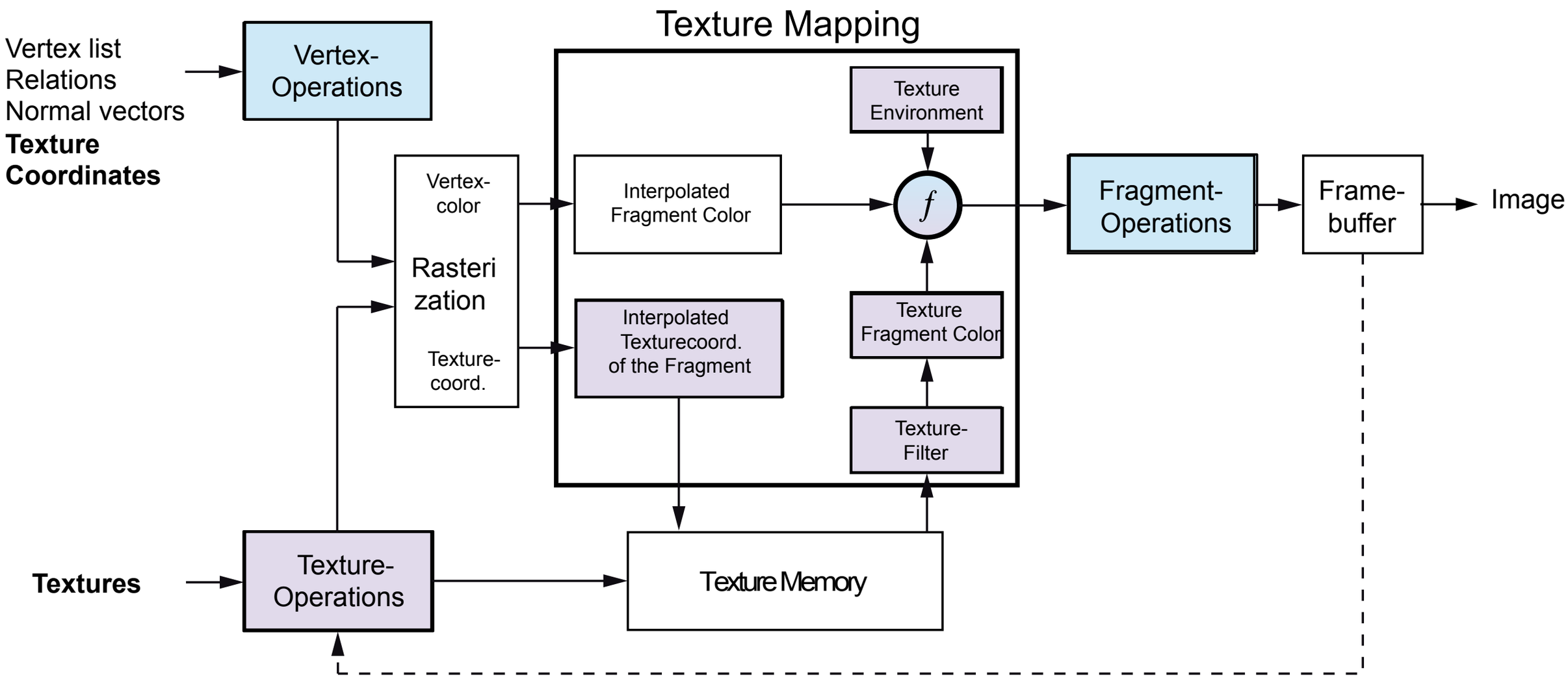
For now: we are talking about photo textures with

- color depth between 8 bit (grey) and 32 bit (color)
- a size of n2 with n = 8, 16, 32, 64
- a color model like RGB, RGBA

Content

- Texture Coordinates
- Texture Parameter
- Texture Environment / Blending
- Texture Filter
- Multi-Texturing

The OpenGL Rendering Architecture



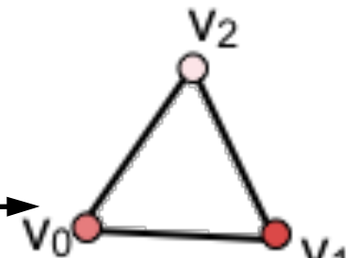
The OpenGL Rendering Pipeline shows the sequence of data processing inside the graphics card

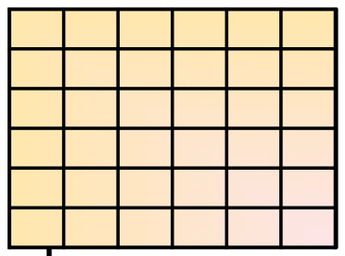
Lighting Operations (last week class)

Texture Mapping

Texture Mapping: get a pixel color

Display List of a 3D model

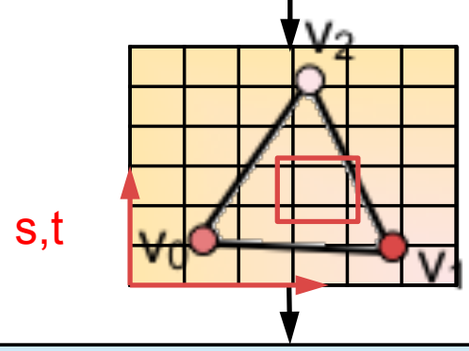




Texture

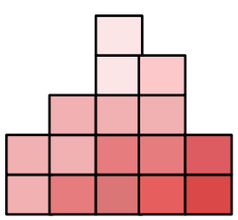
Rasterization

- Interpolation of texture coordinates



Texture Filter

- Approximation of the texture color
- Blending texture color with fragment color

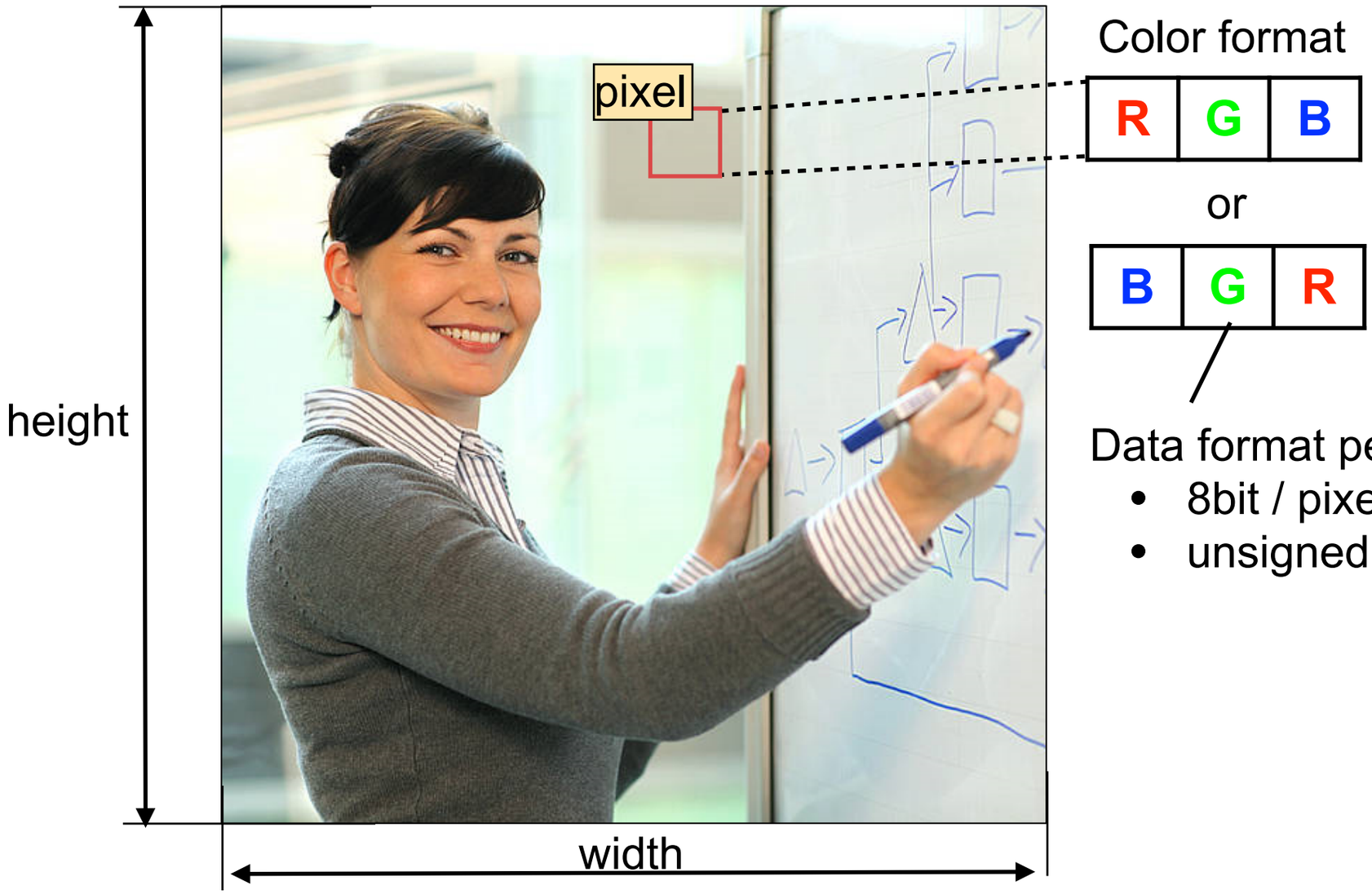


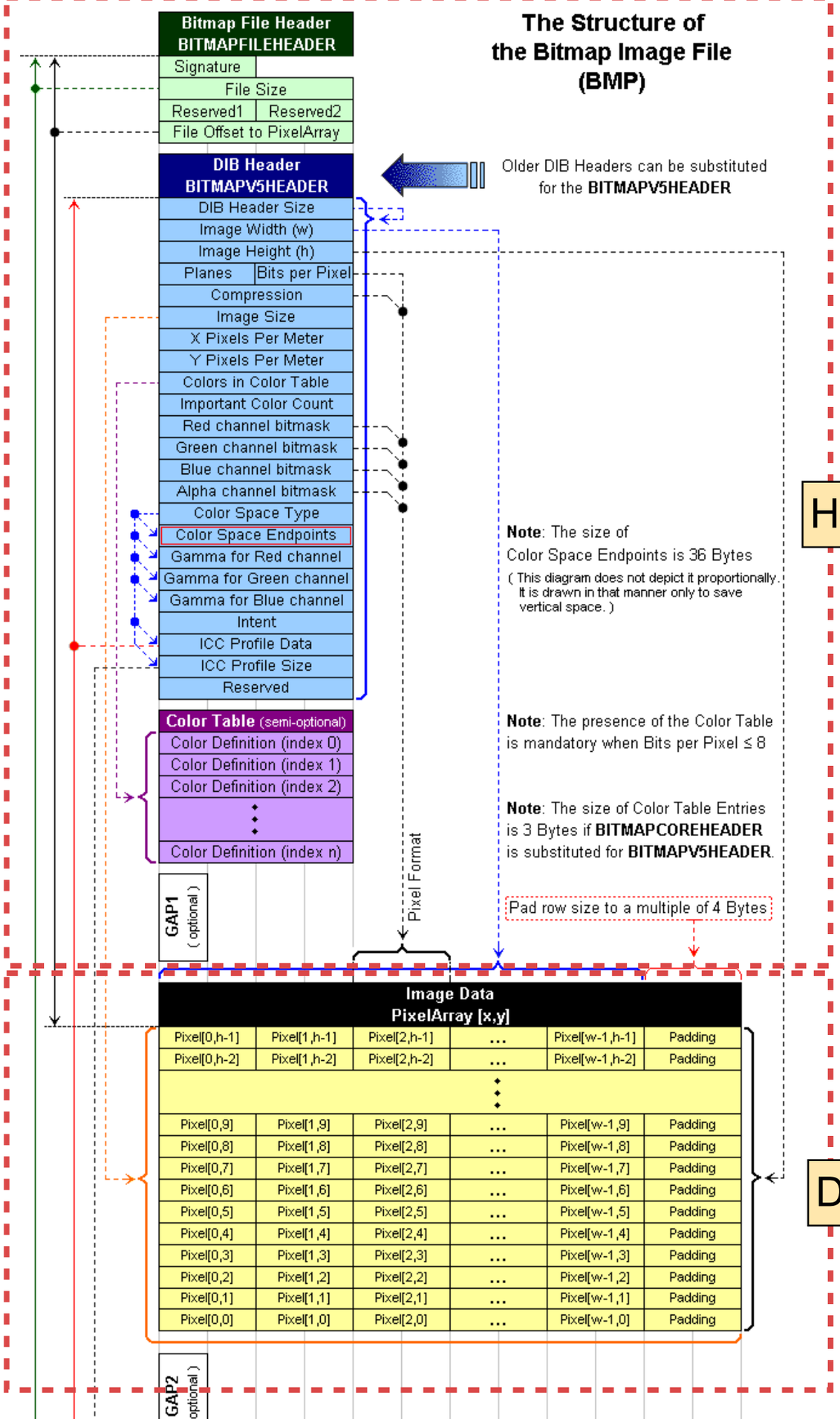
Fragment color

Texture Environment

Image data in Frame Buffer

Texture / Photo





VRAC | HCI

Load an Image

To load an image, you have to open the file and read the data. The file format specification must be considered. Otherwise, you will miss the data.

VRAC|HCI

Load an Image

```
int channels = 3;
unsigned char * data;
unsigned char header[54]; // Each BMP file begins by a 54-bytes header
unsigned int dataPos;     // Position in the file where the actual data begins
unsigned int width, height;
unsigned int imageSize;

// This opens a file
FILE * file;
file = fopen( filename, "rb" );

if ( file == NULL ) return 0;

// This reads the header of the file and checks the length.
if ( fread(header, 1, 54, file)!=54 )
{
    // If not 54 bytes read, this is not a bmp.
    // Only bmp have header of length 54
    printf("Not a correct BMP file\n");
    return false;
}

// Read the start position of the data, the size, the width, and height.
dataPos    = *(int*)&(header[0x0A]);
imageSize  = *(int*)&(header[0x22]);
width      = *(int*)&(header[0x12]);
height     = *(int*)&(header[0x16]);

// Create memory for this texture
data = (unsigned char *)malloc( width * height * channels );

// Read the data from a file.
fread( data, width * height * channels, 1, file );

// Release the file.
fclose( file );
```

Recommendation:
Google for an already existing loader

File Format Specification

Offset #	Size	Purpose
0Eh	4	the size of this header (40 bytes)
12h	4	the bitmap width in pixels (signed integer).
16h	4	the bitmap height in pixels (signed integer).
1Ah	2	the number of color planes being used. Must be set to 1.
1Ch	2	the number of bits per pixel, which is the color depth of the image. Typical values are 1, 4, 8, 16, 24 and 32.
1Eh	4	the compression method being used. See the next table for a list of possible values.
22h	4	the image size. This is the size of the raw bitmap data (see below), and should not be confused with the file size.
26h	4	the horizontal resolution of the image. (pixel per meter, signed integer)
2Ah	4	the vertical resolution of the image. (pixel per meter, signed integer)
2Eh	4	the number of colors in the color palette, or 0 to default to 2^n.
32h	4	the number of important colors used, or 0 when every color is important; generally ignored.

Create a texture

ARLAB

glTexImage2D(GLenum target, GLint level, GLint internalFormat,
GLsizei width, GLsizei height, GLint border, GLenum format,
GLenum type, const GLvoid * data);

Parameters:

- target: the type of texture, e.g., GL_TEXTURE_2D – a 2D texture
- level: the MipMap-Level
- internalFormat: specifies the number of color components and its size that is used on the graphics hardware, e.g., GL_RGB8 says, three components, each 8 bit.
- width of the texture.
- height of the texture.
- border: this value must be 0.
- format, color order of the loaded data. GL_RGB says red, green and blue data in that order.
- type: the type of data, e.g., GL_UNSIGNED_BYTE means the data is unsigned and 1 byte (8bit) describes the color.
- data: bitmapData is the actual bits

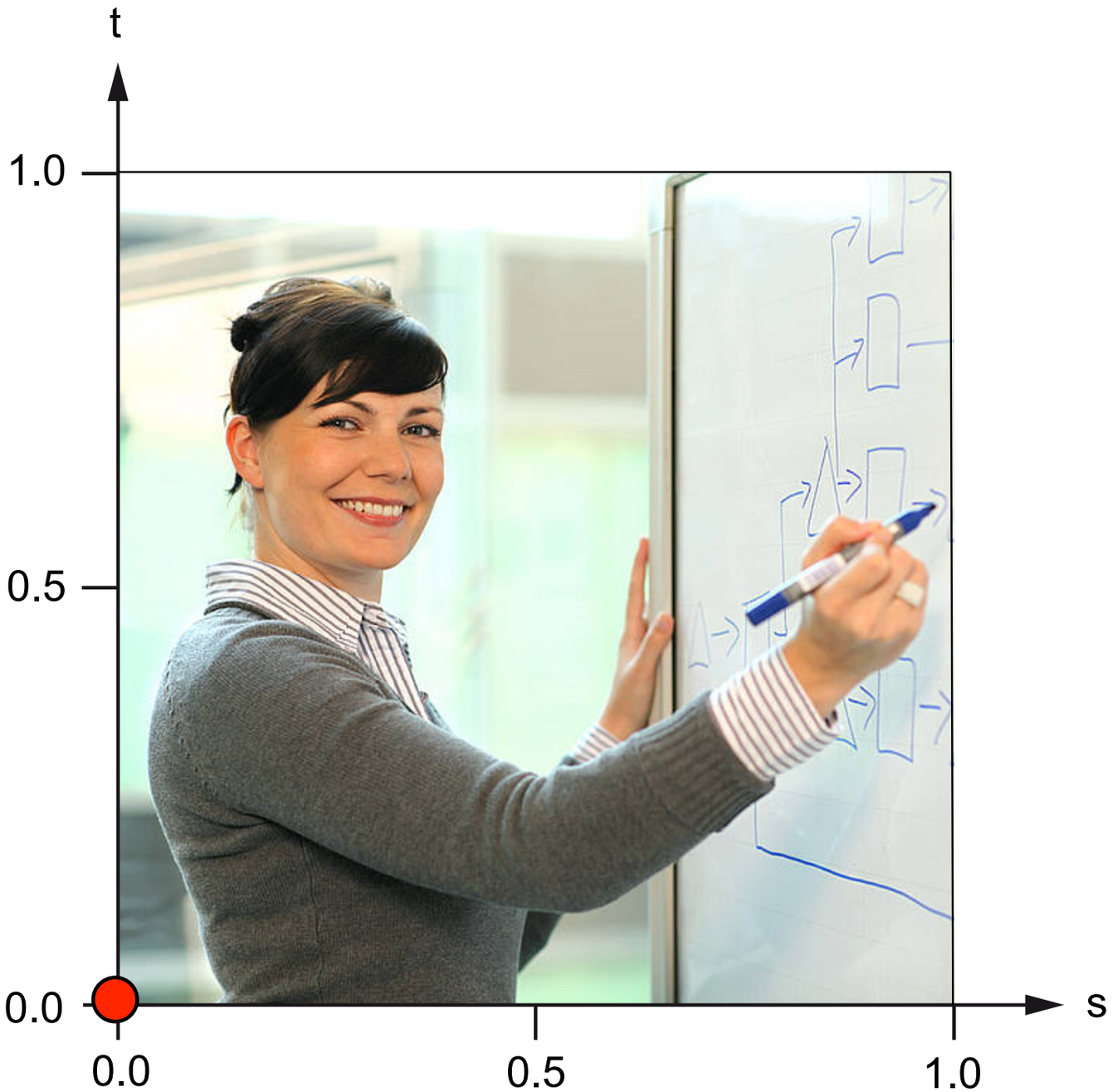
Example

ARLAB

```
[The code to load the data from a file is missing at this location]

// Generate a texture, this function allocates the memory and
 // associates the texture with a variable.
 glGenTextures(0, &texture );

 // Set a texture as active texture.
 glBindTexture( GL_TEXTURE_2D, texture );

 // Change the parameters of your texture units.
   glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,GL_MODULATE );
   glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,GL_NEAREST );
   glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,GL_LINEAR );
   glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,GL_REPEAT );
   glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,GL_REPEAT );
```

```
   // Load the texture to your graphics hardware. This texture is automatically
   // associated with texture 0 and the texture variable "texture”.
   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
      GL_BGR, GL_UNSIGNED_BYTE, data);
```

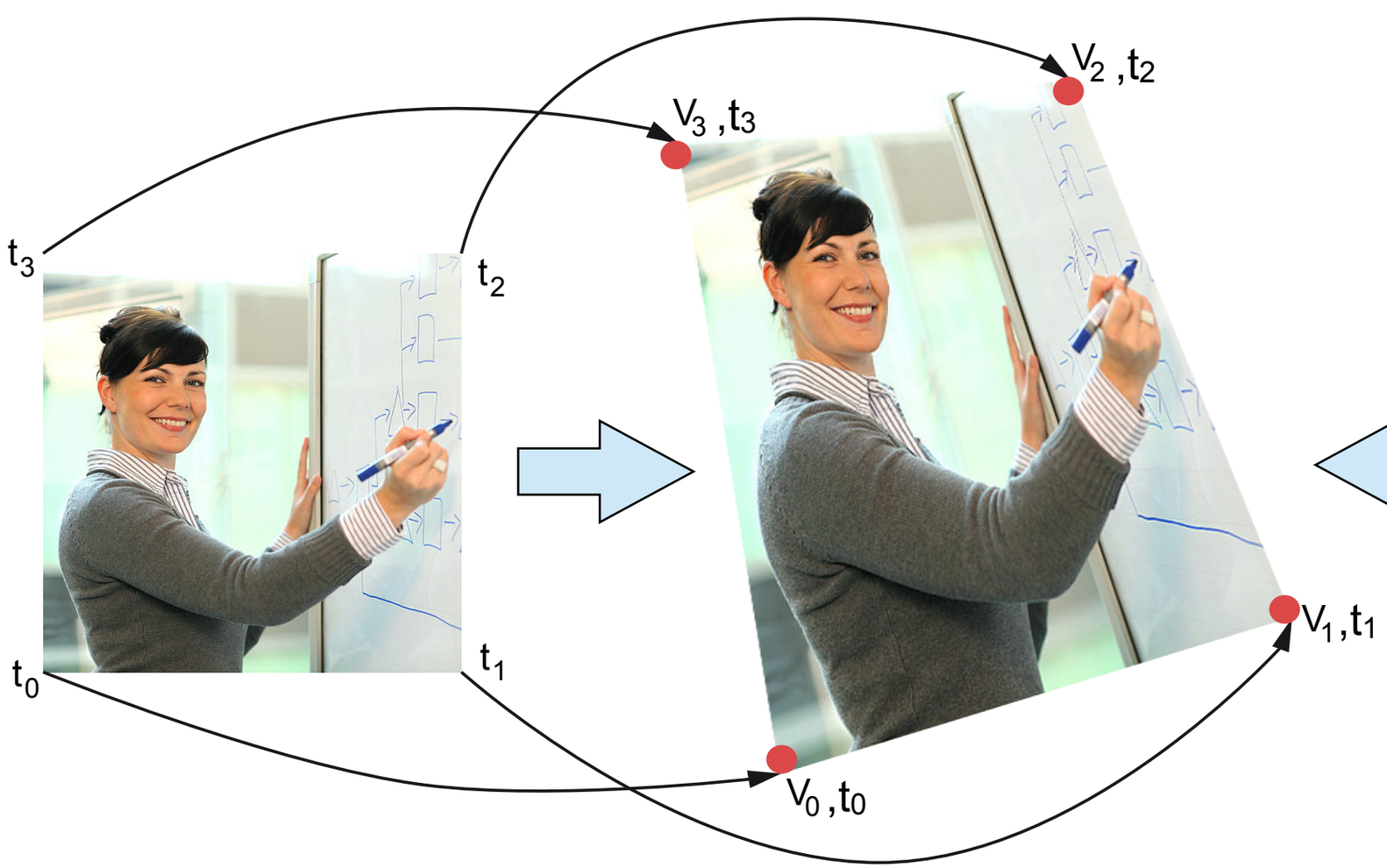
Texture Coordinates (1/2)



A texture can be considered as a 2D image. The position of each texel is described by two texture coordinates (s, t).
By default, the texture coordinates ranges from 0 to 1.

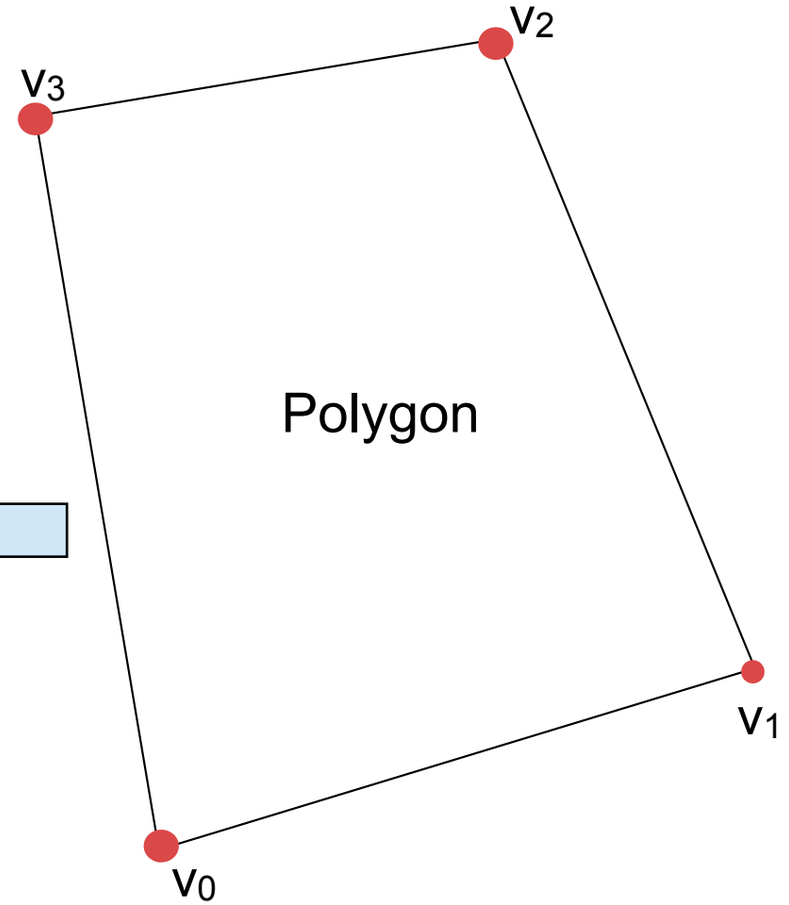
Texture Coordinates (2/2)

A texture coordinate specifies the Texel of a texture that is mapped to a distinct vertex.



A Texture with image coordinates t_i = (u,v)

The association between the t_i and v_i (finally) results in a texture mapped on the surface



A surface (quad) with four vertices v_i

Texture Coordinates in OpenGL

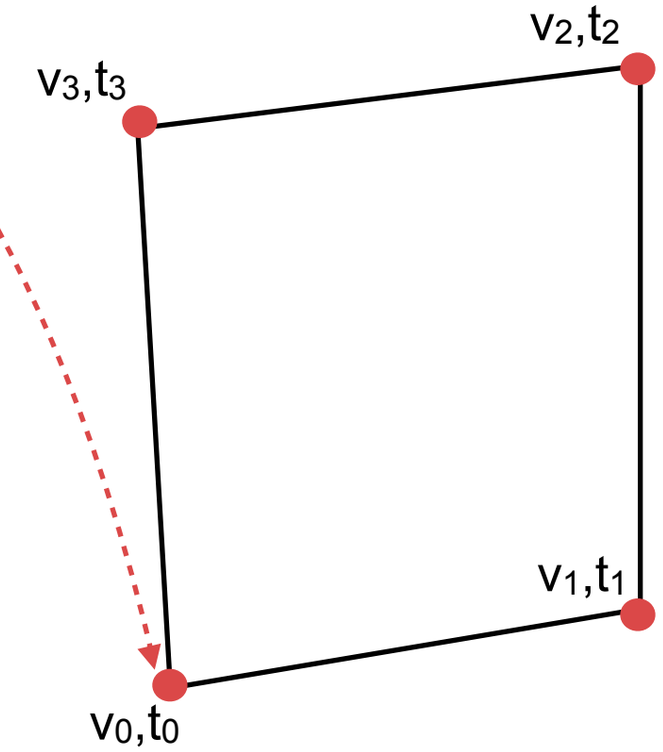
Texture Coordinate ⟷ **Vertex**

Associates a point in texture space (s,t) with a vertex of the polygon face

Example

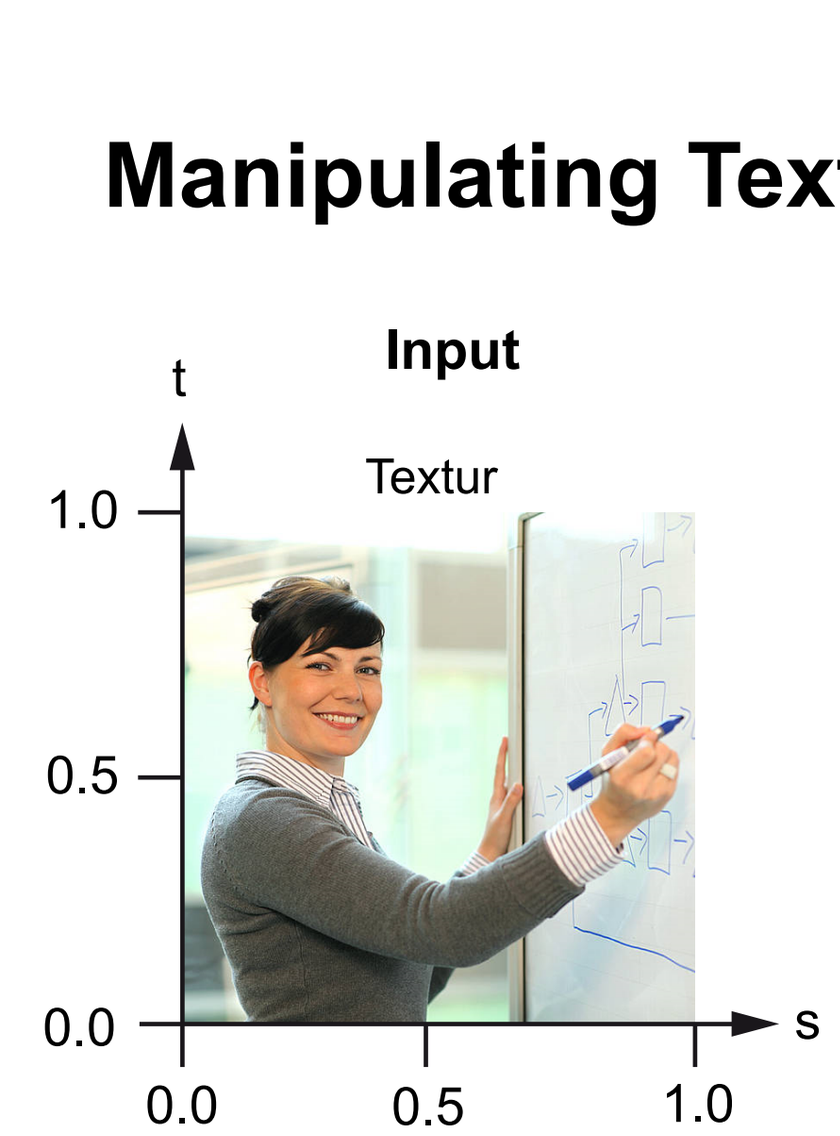
```
GLfloat vertexData[] = {
    //  X     Y     Z       U     V
      0.0f, 0.0f, 0.0f,   0.0f, 0.0f,
      0.0f, 0.8f, 0.0f,   0.5f, 0.0f,
      0.8f, 0.8f, 0.0f,   1.0f, 1.0f,
      0.8f, 0.0f, 0.0f,   0.0f, 0.5f,
};
```

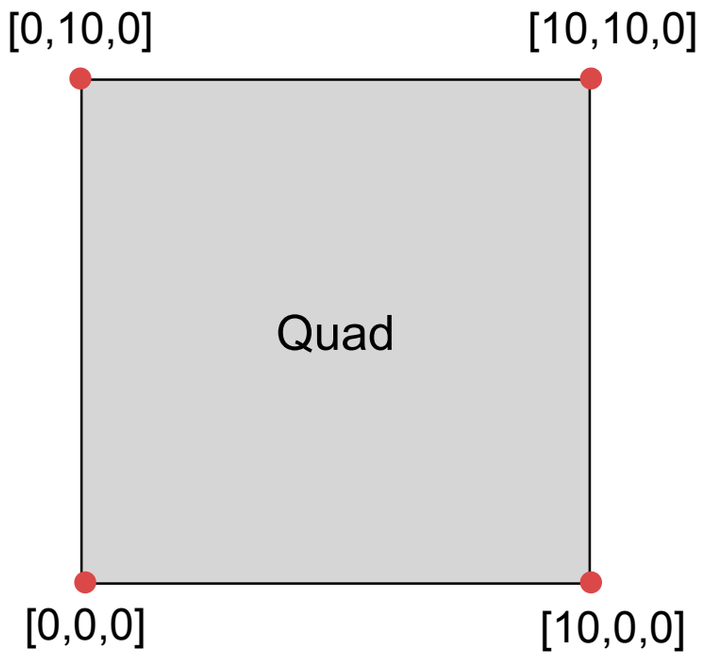
Association is established by coding a texture coordinate just before creating the vertex

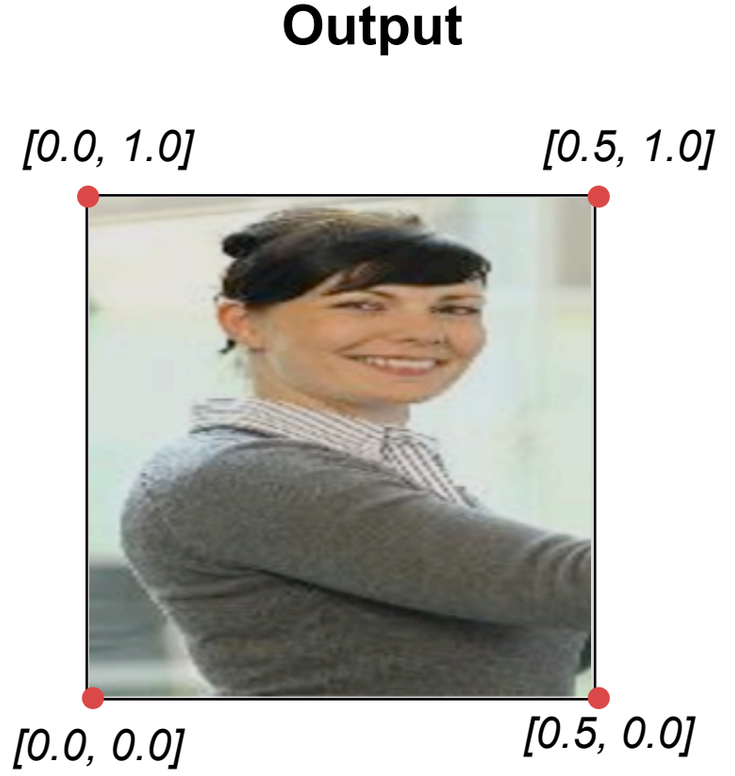


The texture coordinates belong to the OpenGL primitive. They are defined regardless of a texture (the programmer should know the texture). Whatever texture will come, the pixel with the texture coordinate t_x is tied to the vertex v_x.

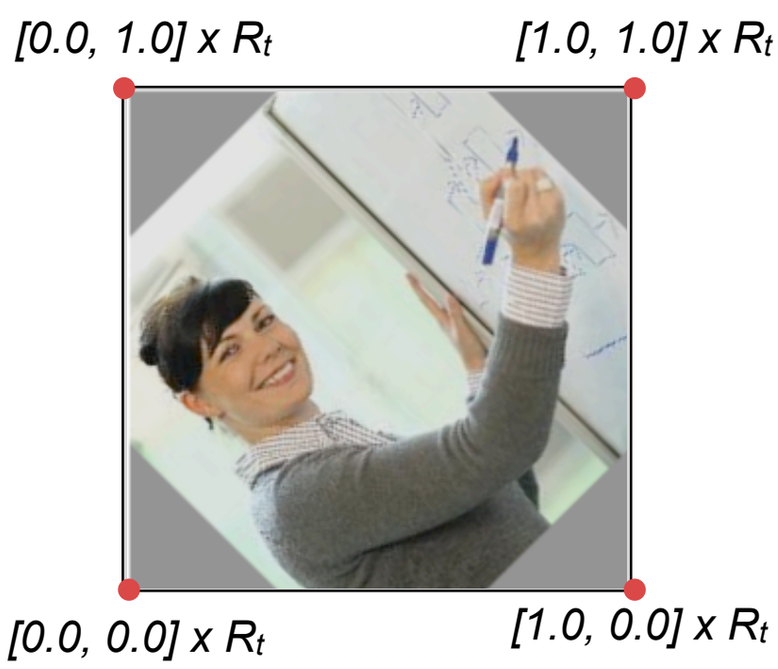
Manipulating Texture Coordinates








```
GLfloat vertexData[] = {
    //U     V
     0.0f, 0.0f,
     0.5f, 0.0f,
     0.5f, 1.0f,
     0.0f, 1.0f,
};
```



$$R_t = \begin{bmatrix} cos(\phi) & -sin(\phi) \\ sin(\phi) & cos(\phi) \end{bmatrix}$$

Sending points to the Shader

ARLAB

This shows how we have done this before when we only used vertices.

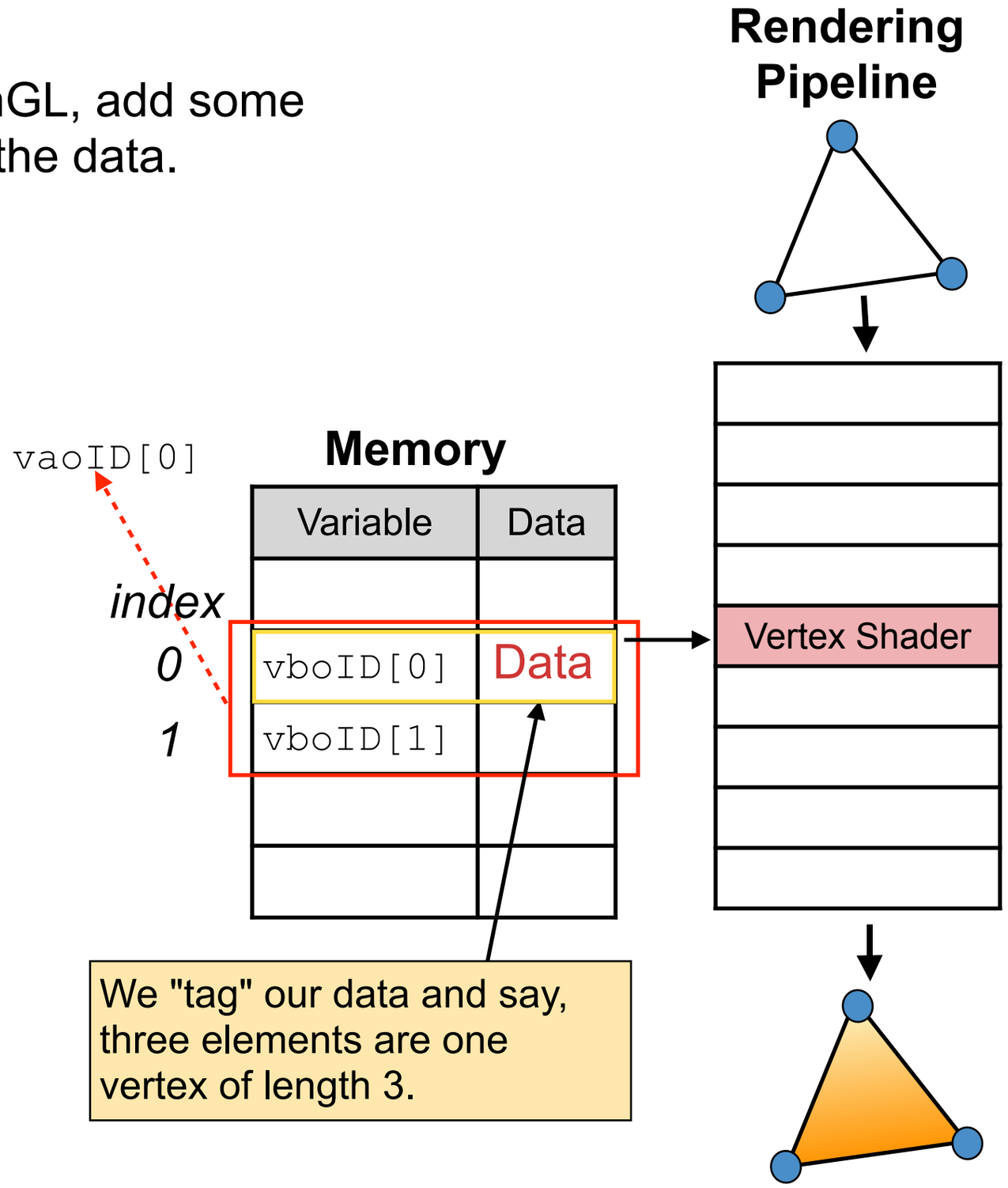
```
// Put the three triangle vertices into the VBO
GLfloat vertexData[] = {
    //  X     Y     Z
     0.0f, 0.0f, 0.0f,
      0.0f, 0.8f, 0.0f,
      0.8f, 0.8f, 0.0f,
      0.8f, 0.0f, 0.0f
};

// connect the xyz to the "vert" attribute of the vertex shader
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, NULL);
```

Define the Buffer Data

Explain OpenGL what the data is about.

Note, these are only 0,1 values for OpenGL, add some meaning that allow OpenGL to interpret the data.

```
glVertexAttribPointer((GLuint)0, 3,
                      GL_FLOAT,
                      GL_FALSE,
                      0, 0);
```

Define the Buffer Data

ARLAB

```
void glVertexAttribPointer( GLuint index, GLint size, GLenum type,
GLboolean normalized, GLsizei stride, const GLvoid * pointer);
```

Define an array of generic vertex attribute data or in other words, explain the meaning of your variables.

Parameters:

- index: Specifies the index of the generic vertex attribute to be modified.
- size: Specifies the number of components per generic vertex attribute. Must be 1, 2, 3, 4. Additionally, the symbolic constant GL_BGRA is accepted by glVertexAttribPointer. The initial value is 4.
- type: Specifies the data type of each component in the array. The symbolic constants GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT, and GL_UNSIGNED_INT, GL_FLOAT, GL_DOUBLE, GL_FIXED
- normalized: values should be normalized (GL_TRUE) or not (GL_FALSE)
- **stride: Specifies the byte offset between consecutive generic vertex attributes.**
- **pointer: Specifies an offset of the first component of the first generic vertex attribute in the array in the data store of the buffer currently bound to the GL_ARRAY_BUFFER target.**

Sending points to the Shader


```
GLfloat vertexData[] = {
    //  X      Y      Z       U      V
     0.0f, 0.0f, 0.0f,   0.0f, 0.0f,
      0.0f, 0.8f, 0.0f,   0.5f, 0.0f,
     0.8f, 0.8f, 0.0f,   1.0f, 1.0f,
     0.8f, 0.0f, 0.0f,   0.0f, 0.5f
};
```

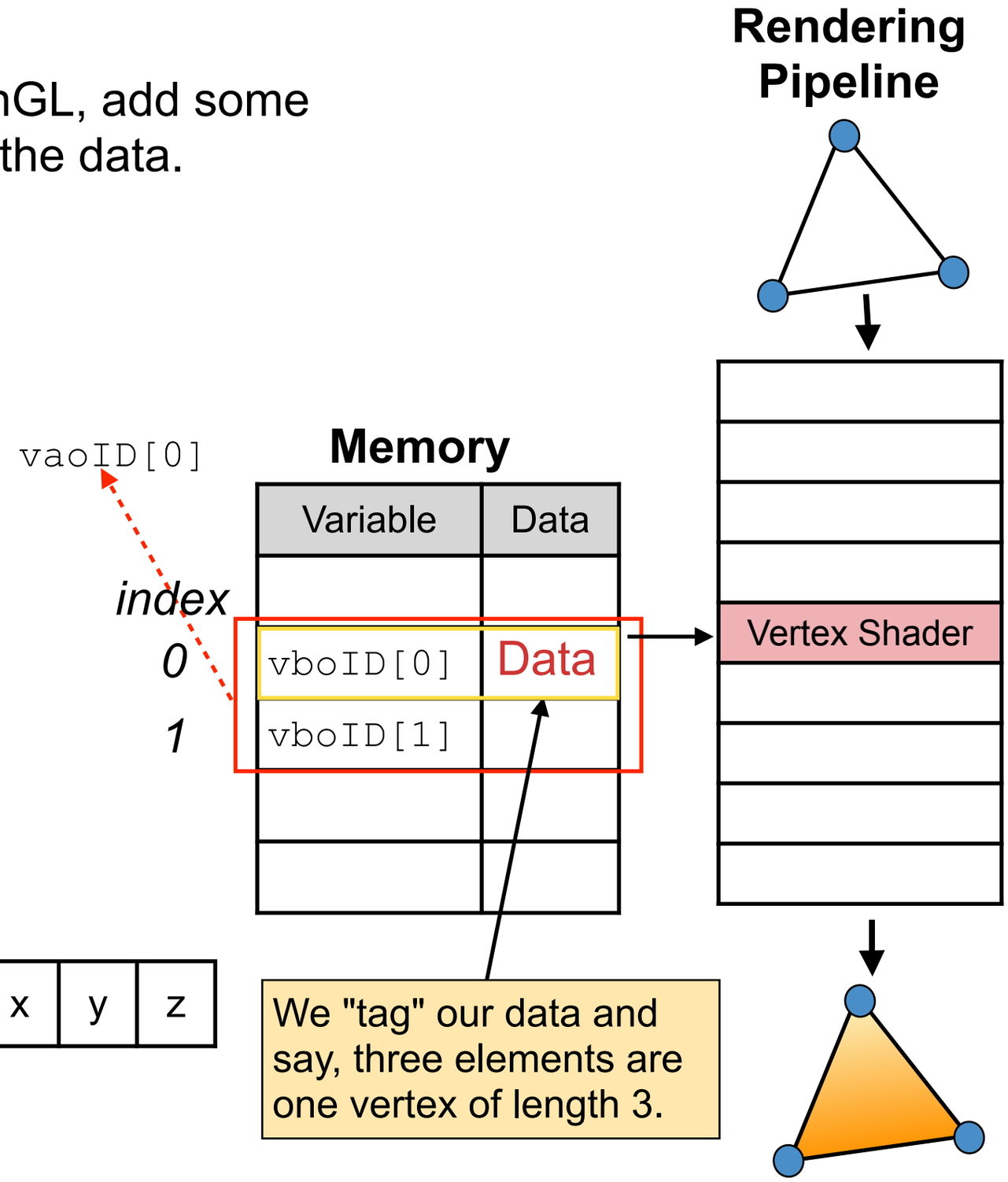
```
// connect the xyz to the "vert" attribute of the vertex shader
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 5*sizeof(GLfloat), NULL);

// connect the uv coords to the "vertTexCoord" attribute of the vertex
shader
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 2, GL_FLOAT, GL_TRUE,  5*sizeof(GLfloat),
                      (const GLvoid*)(3 * sizeof(GLfloat)));
```

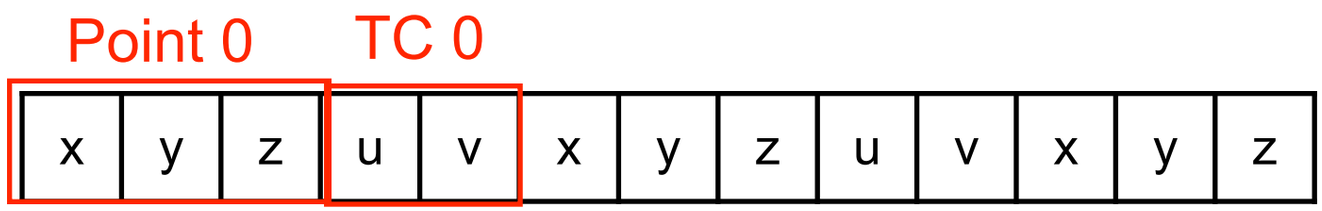
Define the Buffer Data

Explain OpenGL what the data is about.

Note, these are only 0,1 values for OpenGL, add some meaning that allow OpenGL to interpret the data.

```
glVertexAttribPointer((GLuint)0, 3,
                      GL_FLOAT,
                      GL_FALSE,
                      0, 0);
```

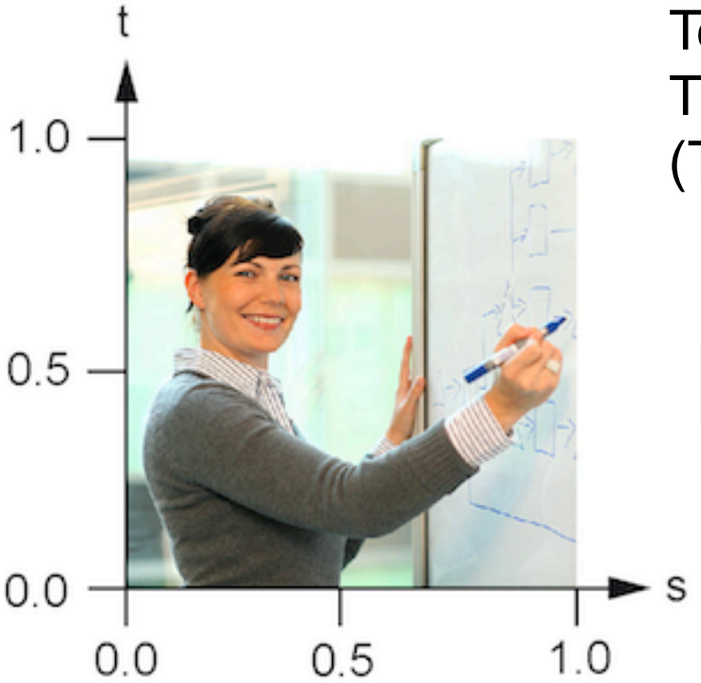




length = 3

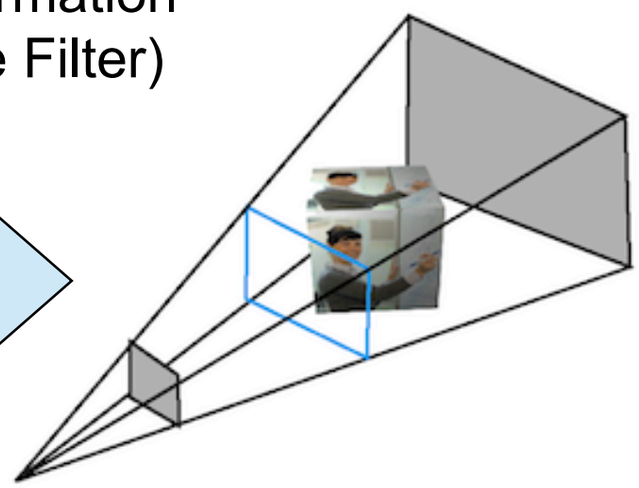
TC: texture coordinate

The Texture Rendering Process



Texture

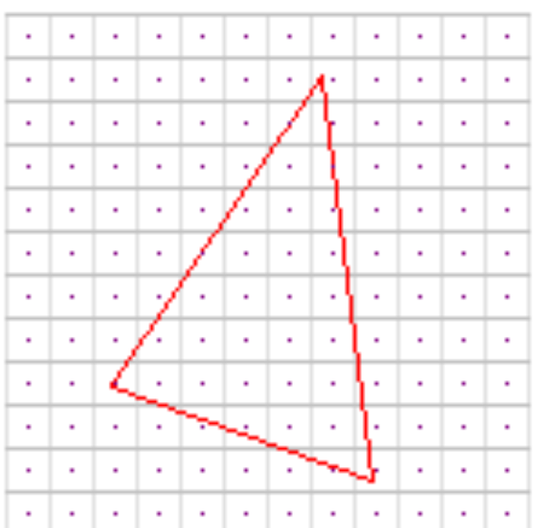
Texture Object Transformation (Texture Filter)

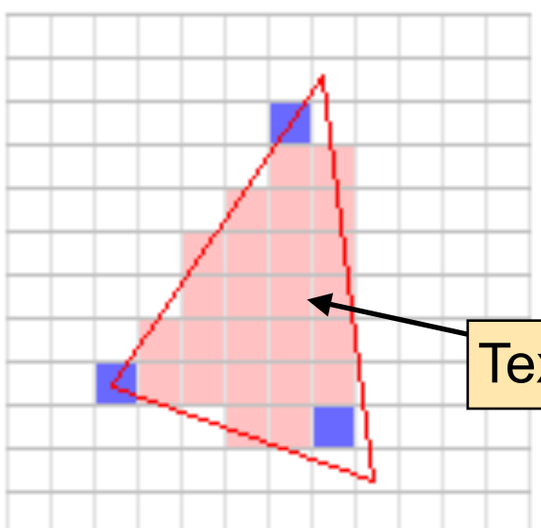


Texture mapped to a object and perspective distorted

Screen Transformation

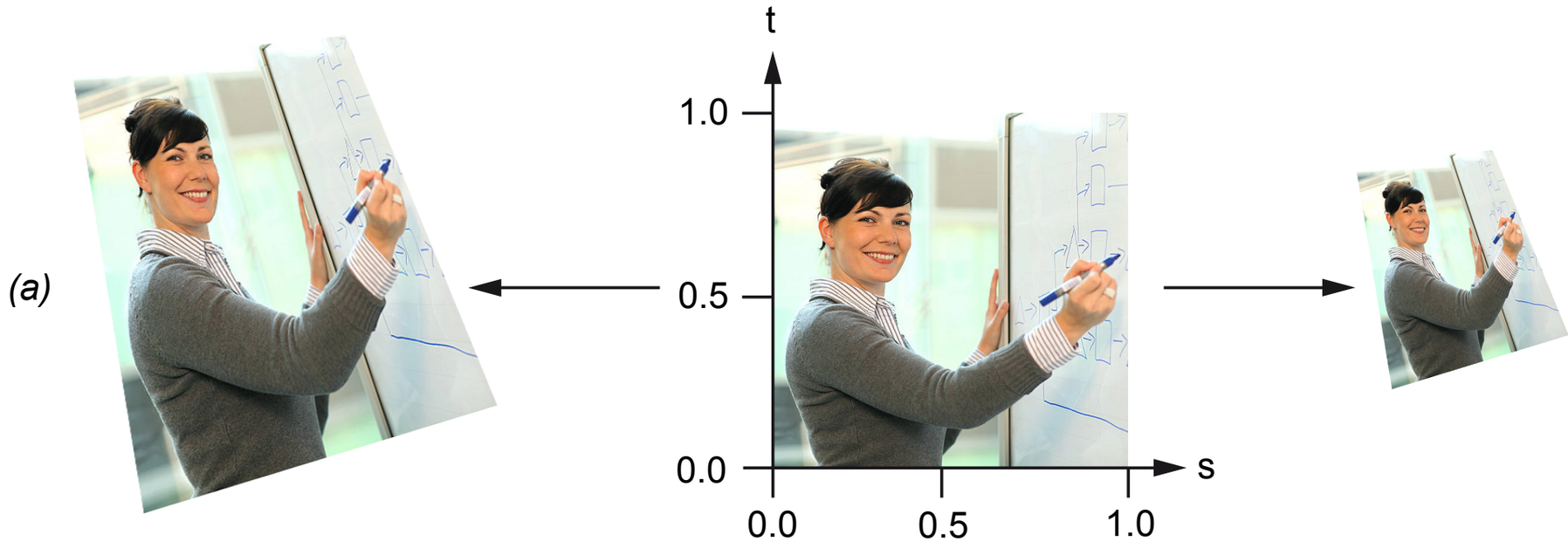
The output on screen





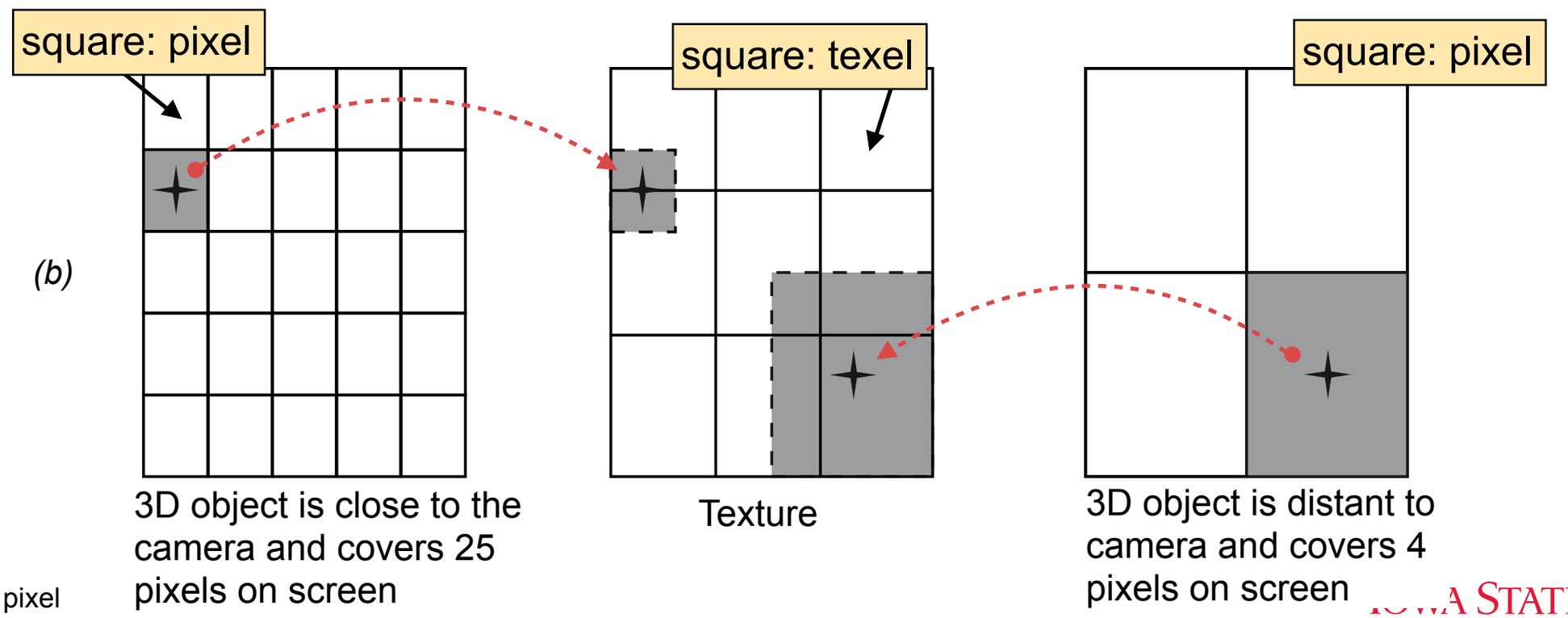
The graphics card must identify the correct color for each fragment (pixel) of a primitive.

Magnification & Minifying Function

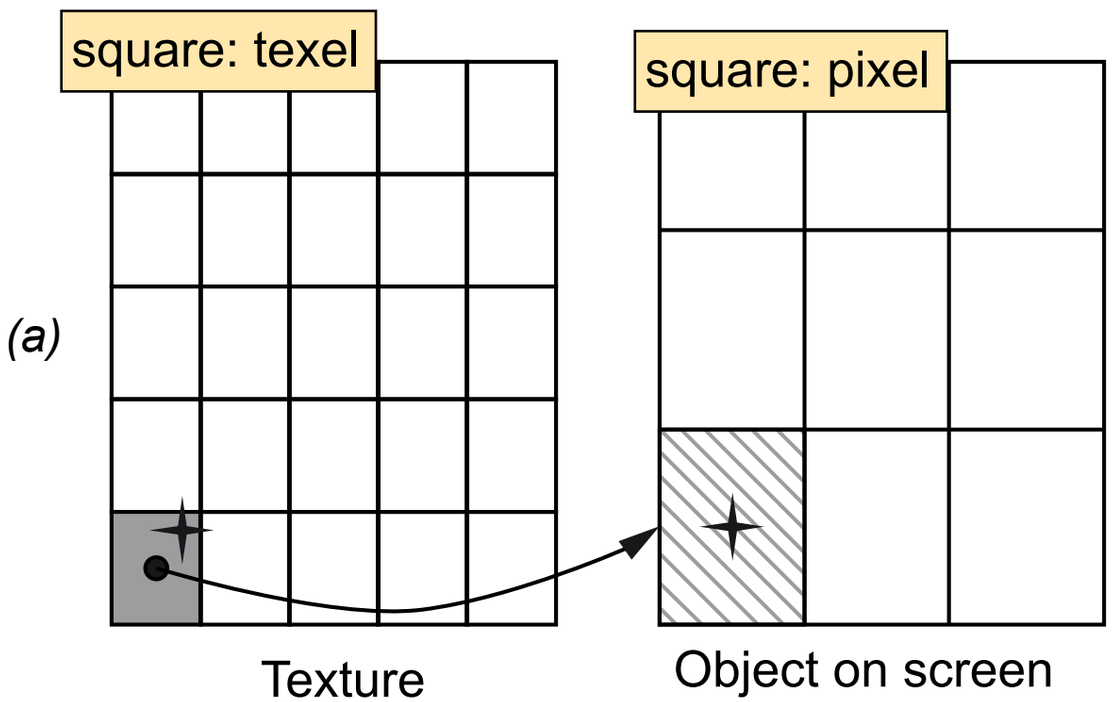


Texture Magnification Function

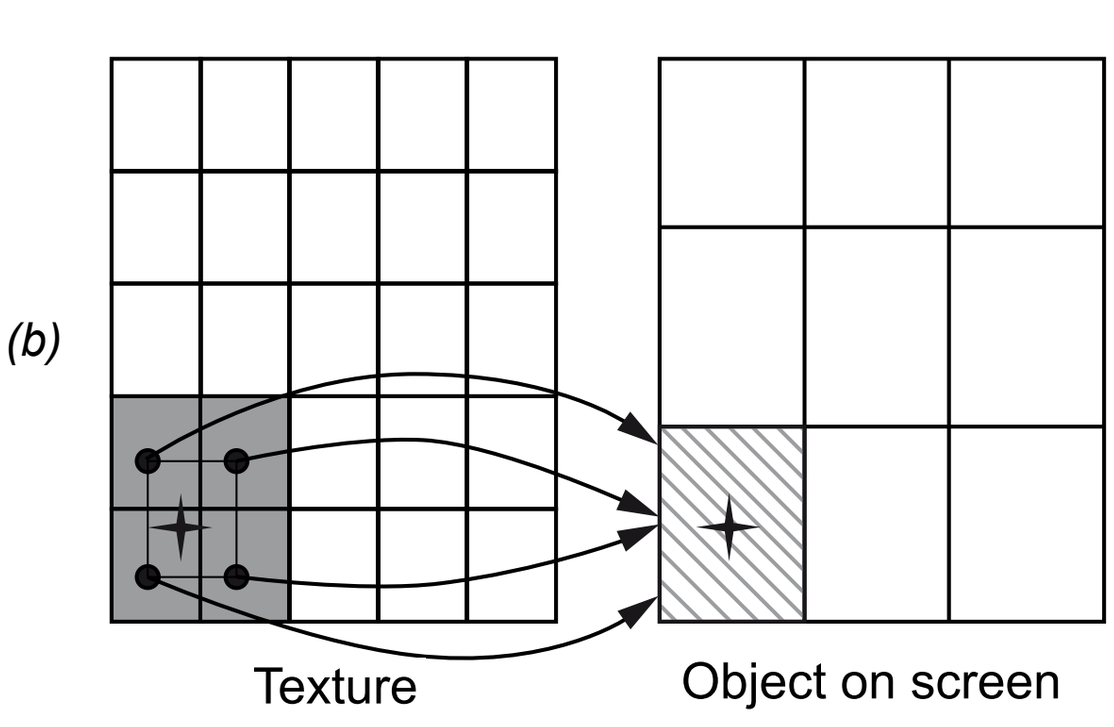
Texture Minifying Function



Texture Filter (1/2)



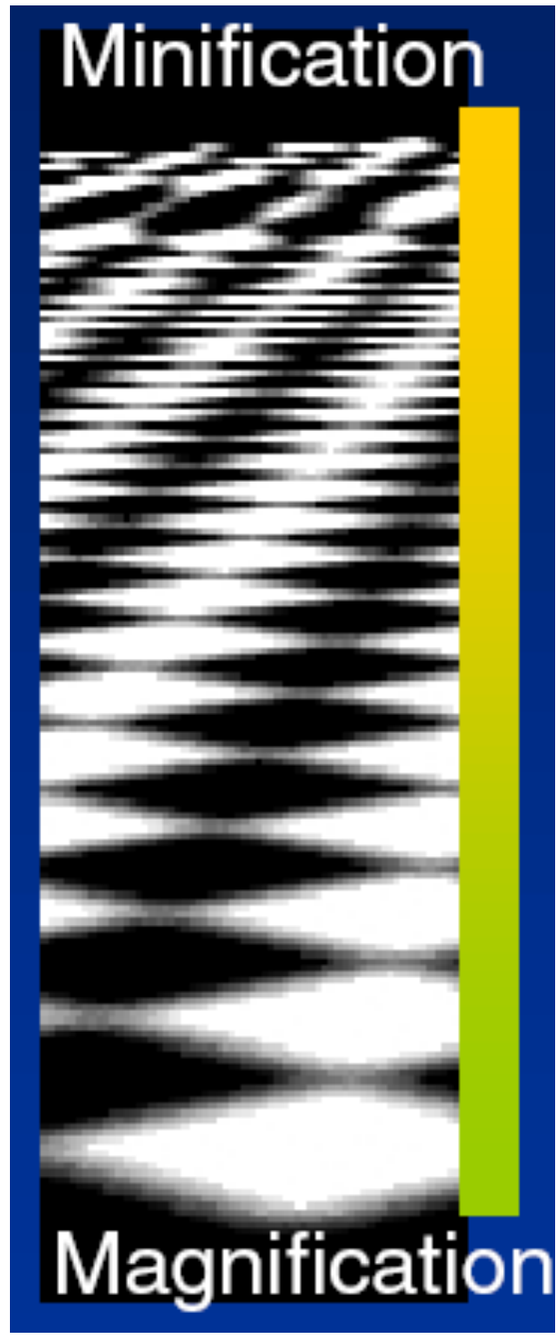
Point Sampling (GL_NEAREST)
The color value closest to the center of the pixel is selected. This is the fastest solution, but it causes aliasing.



Bilinear Sampling (GL_LINEAR)
The average value of the four (2x2) closest texels is selected. This operation is slower but results in smoother rendering.

Texture Filter (2/2)

- The texture **magnification function** is used when the pixel being textured maps to an area less than or equal to one texture element.
- The **texture minifying function** is used whenever the pixel being textured maps to an area greater than one texture element.
- Both filters can be specified at the same time. The graphics card applies them to one object at the same time, e.g., the front of the object is close to the virtual camera and the back of the object is somewhere in the background.



Texture Filter

glTexParameterf (GLenum target, GLenum pname, GLfloat param)

OpenGL function to setup a specific parameter

- target: Specifies the target texture, which must be either GL_TEXTURE_1D, GL_TEXTURE_2D, or GL_TEXTURE_3D.
- pname: Specifies the symbolic name of a single-valued texture parameter. pname can be one of the following:
  - GL_TEXTURE_MAG_FILTER
    to change the magnification filter function
  - GL_TEXTURE_MIN_FILTER
    to change the minifying filter function
- param: Specifies the value of pname like:
  - GL_LINEAR
    Interpolates the color of a fragment.
    makes the texture look smooth way in the distance, and when it's up close to the screen, but is expensive,
  - GL_NEAREST
    Uses the closes color information available in the texture.

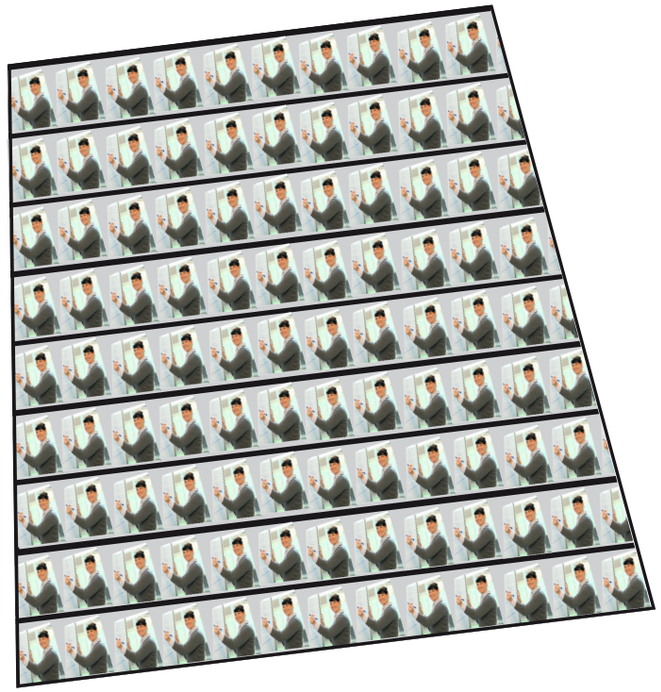
Example

ARLAB

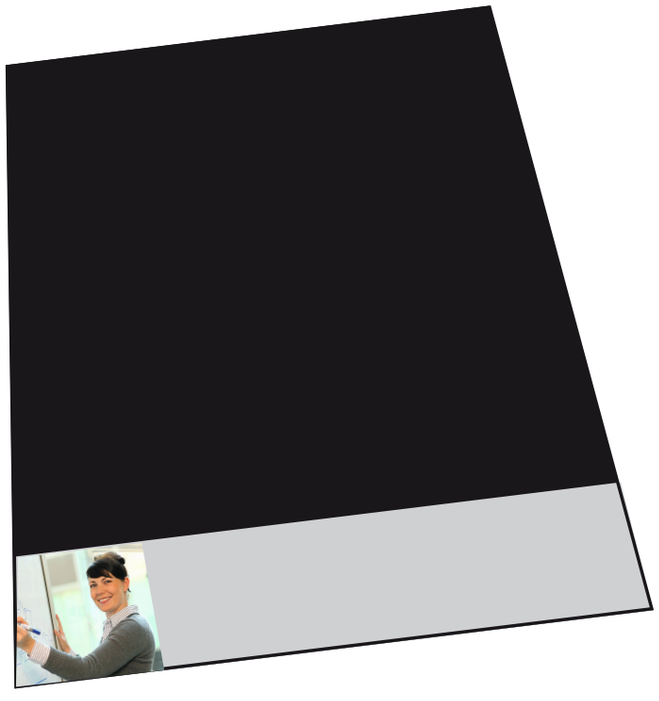
```
[The code to load the data from a file is missing at this location]

// Generate a texture, this function allocates the memory and
 // associates the texture with a variable.
 glGenTextures(0, &texture );

 // Set a texture as active texture.
 glBindTexture( GL_TEXTURE_2D, texture );

 // Change the parameters of your texture units.
 glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,GL_MODULATE );
 glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,GL_NEAREST );
 glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,GL_LINEAR );

 glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,GL_REPEAT );
 glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,GL_REPEAT );

 // Load the texture to your graphics hardware. This texture is automatically
 // associated with texture 0 and the texture variable "texture”.
 glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
    GL_BGR, GL_UNSIGNED_BYTE, data);
```

Texture Wraps

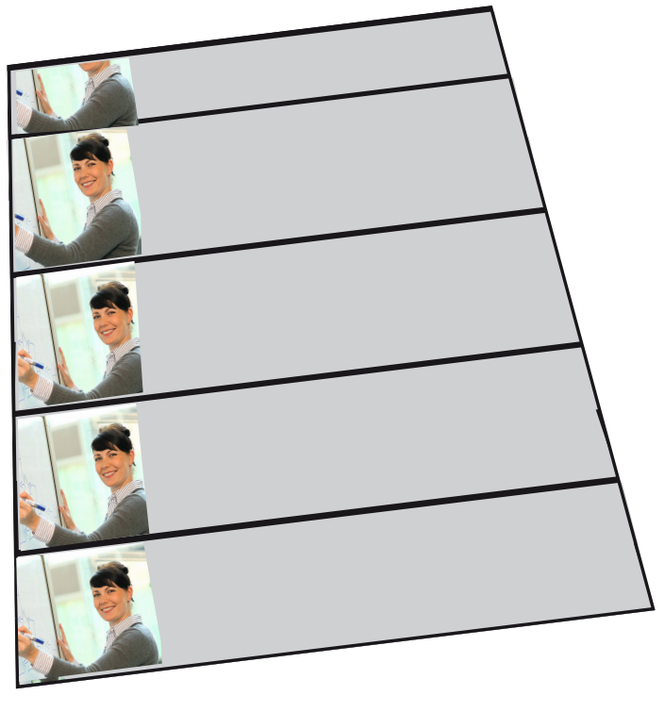
Texture



GL_REPEAT



GL_CLAMP in x-direction



GL_CLAMP / GL_REPEAT

In some cases, surfaces with regular patterns need to be created. Therefore, OpenGL provides texture wrap operations, which continue the texture image automatically.

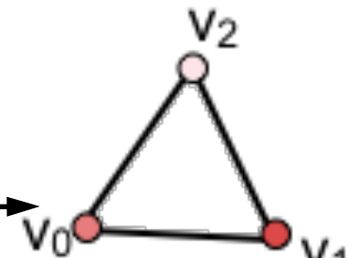
GL_REPEAT: The texture is applied multiple times until the entire object is covered.

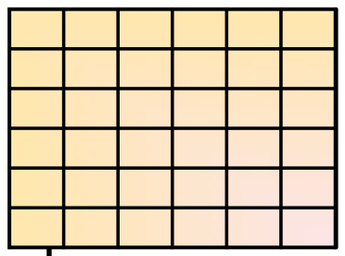
GL_CLAMP: The last column / row of a texture is repeated until the entire object is covered.

Both parameter can be specified individual for x and y direction.

Texture Mapping: get a pixel color

Display List of a 3D model

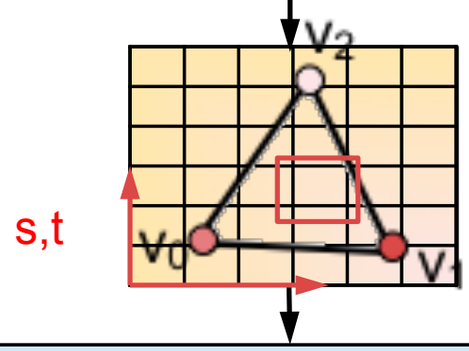




Texture

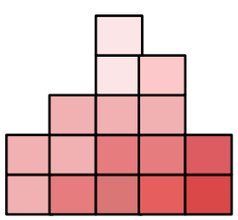
Rasterization

- Interpolation of texture coordinates



Texture Filter

- Approximation of the texture color
- Blending texture color with fragment color



Fragment color

Texture Environment

Image data in Frame Buffer

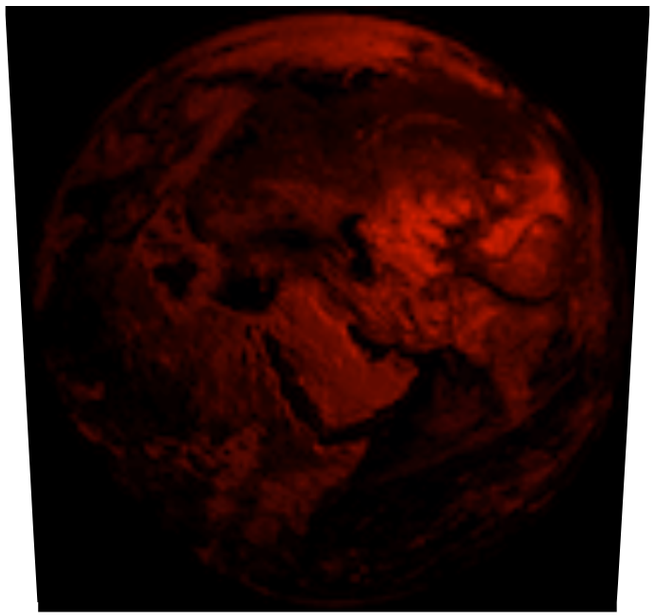
Texture Environment / Blending
GL_MODULATE


Primitive

+


Texture (GL_RGB)

=


Result

```
glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,GL_MODULATE );
```

Internal format: GL_RGB

$$\begin{bmatrix} R_f \cdot R_t \\ G_f \cdot G_t \\ B_f \cdot B_t \\ A_f \end{bmatrix} = \begin{bmatrix} R_o \\ G_o \\ B_o \\ A_o \end{bmatrix}$$

Internal format: GL_RGBA

$$\begin{bmatrix} R_f \cdot R_t \\ G_f \cdot G_t \\ B_f \cdot B_t \\ A_f \cdot A_t \end{bmatrix} = \begin{bmatrix} R_o \\ G_o \\ B_o \\ A_o \end{bmatrix}$$

R_f, G_f, B_f : face (primitive color)
R_t, G_t, B_t : texture color

Texture Environment / Blending

GL_BLEND


```
glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,GL_BLEND );
```

Internal format: GL_RGB

$$\begin{bmatrix} (1-R_t)R_f + R_T R_C \\ (1-G_t)G_f + G_T R_C \\ (1-B_t)B_f + B_T B_C \\ A_f \end{bmatrix} = \begin{bmatrix} R_o \\ G_o \\ B_o \\ A_o \end{bmatrix}$$

Internal format: GL_RGBA

$$\begin{bmatrix} (1-R_t)R_f + R_T R_C \\ (1-G_t)G_f + G_T R_C \\ (1-B_t)B_f + B_T B_C \\ A_f \cdot A_t \end{bmatrix} = \begin{bmatrix} R_o \\ G_o \\ B_o \\ A_o \end{bmatrix}$$

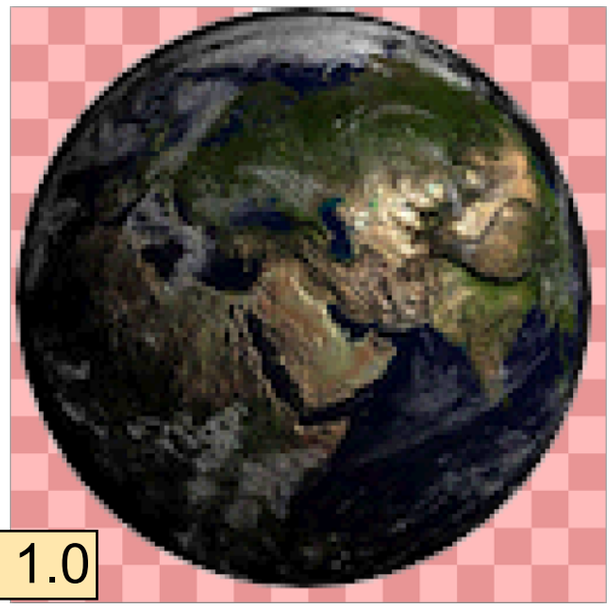
R_f, G_f, B_f : face (primitive color)
R_t, G_t, B_t : texture color
R_c, G_c, B_c : blend color

Texture Environment / Blending

GL_DECAL

Primitive

+



Texture (GL_RGB**A**)

=

Result

```
glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,GL_DECAL );
```

Internal format: GL_RGB

$$\begin{bmatrix} (1-A_t)R_f + R_T R_C \\ (1-A_t)G_f + G_T R_C \\ (1-A_t)B_f + B_T B_C \\ A_f \end{bmatrix} = \begin{bmatrix} R_o \\ G_o \\ B_o \\ A_o \end{bmatrix}$$

Internal format: GL_RGBA

$$\begin{bmatrix} (1-A_t)R_f + R_T R_C \\ (1-A_t)G_f + G_T R_C \\ (1-A_t)B_f + B_T B_C \\ A_f \cdot A_t \end{bmatrix} = \begin{bmatrix} R_o \\ G_o \\ B_o \\ A_o \end{bmatrix}$$

R_f, G_f, B_f : face (primitive color)
R_t, G_t, B_t : texture color
R_c, G_c, B_c : blend color

Texture Environment / Blending

The Texture Environment unit of the graphics card is responsible for the blending of the texture color with the background color.

```
void glTexEnvf(GLenum target, GLenum pname, GLfloat param);
```

- target: Specifies a texture environment; we need GL_TEXTURE_ENV.
- pname: Specifies the symbolic name of a single-valued texture environment parameter; we need GL_TEXTURE_ENV_MODE
- param: Specifies a single symbolic constant: GL_BLEND, GL_DECAL, GL_REPLACE, GL_MODULATE

The calculation (blending of colors) depends on the internal texture format

```
glTexImage2D(  GLenum  target, GLint level, GLint internalFormat,
               GLsizei  width,  GLsizei  height, GLint  border, GLenum  format,
               GLenum  type,  const GLvoid *  data);
```

Thank you!

Questions

Rafael Radkowski, Ph.D.
Iowa State University
Virtual Reality Applications Center
1620 Howe Hall
Ames, Iowa 5001, USA

+1 515.294.5580
+1 515.294.5530(fax)

rafael@iastate.edu

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY