ARLAB

ME/CprE/ComS 557

Computer Graphics and Geometric Modeling

Texture Mapping with OpenGL

October 13th, 2015

Rafael Radkowski

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

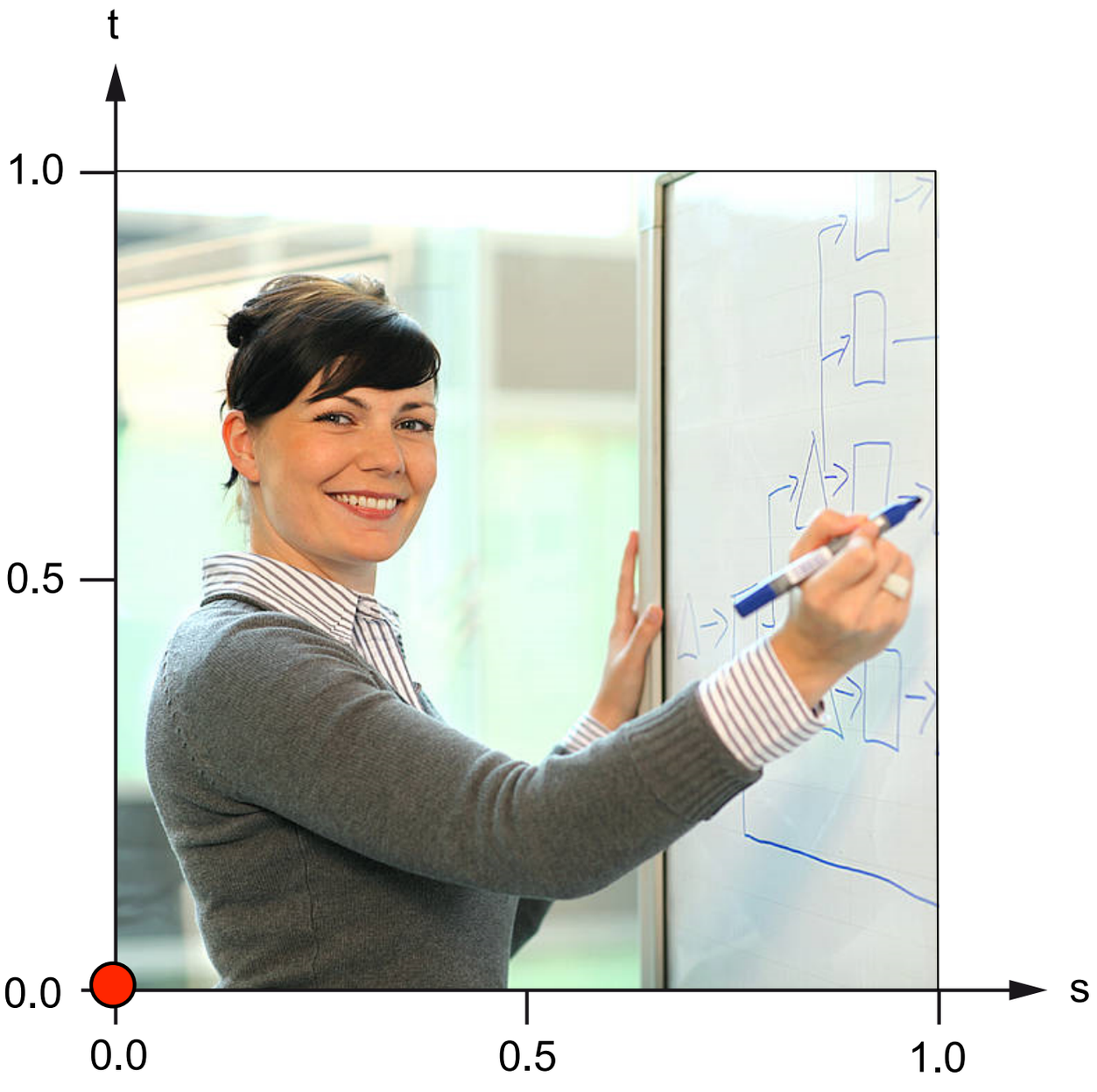
Content

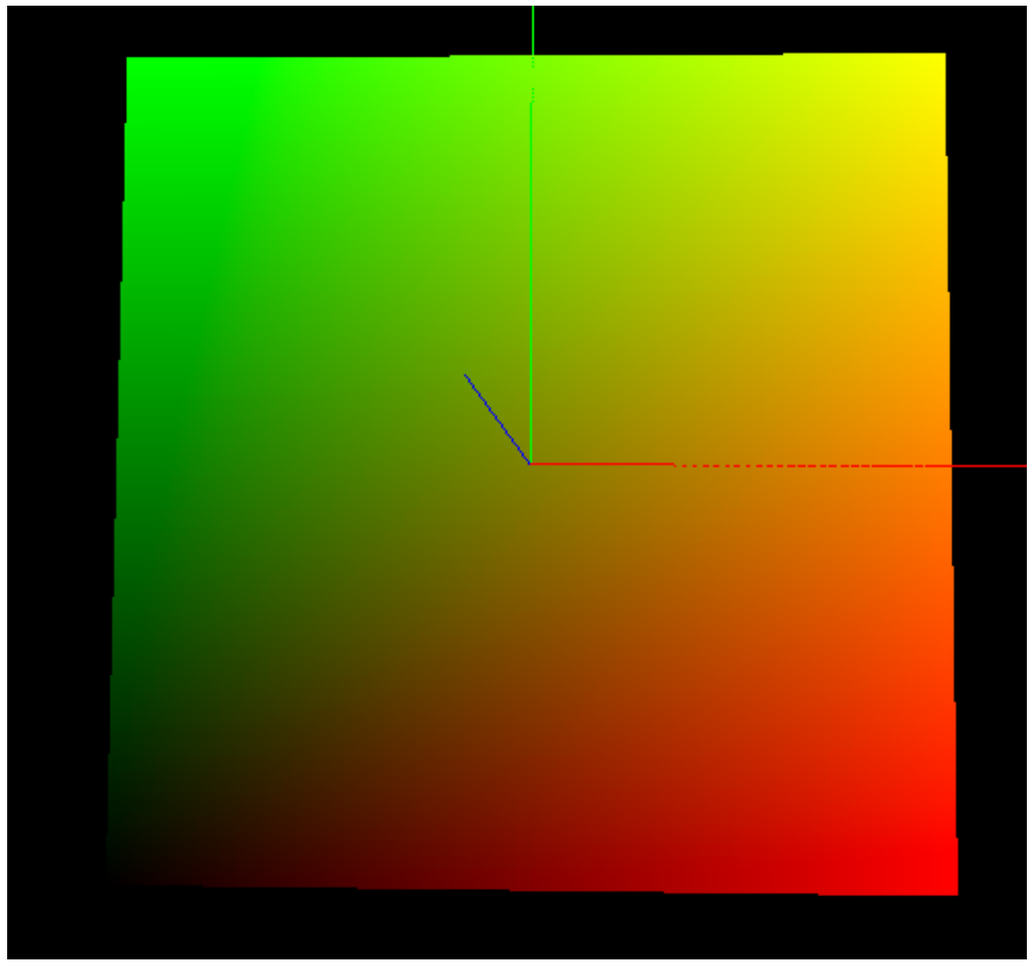
- Create a texture
- Create a uniform variable
- Map the texture to the primitive

We already know

ARLAB

how to define texture coordinates and how we associate them with a primitive.

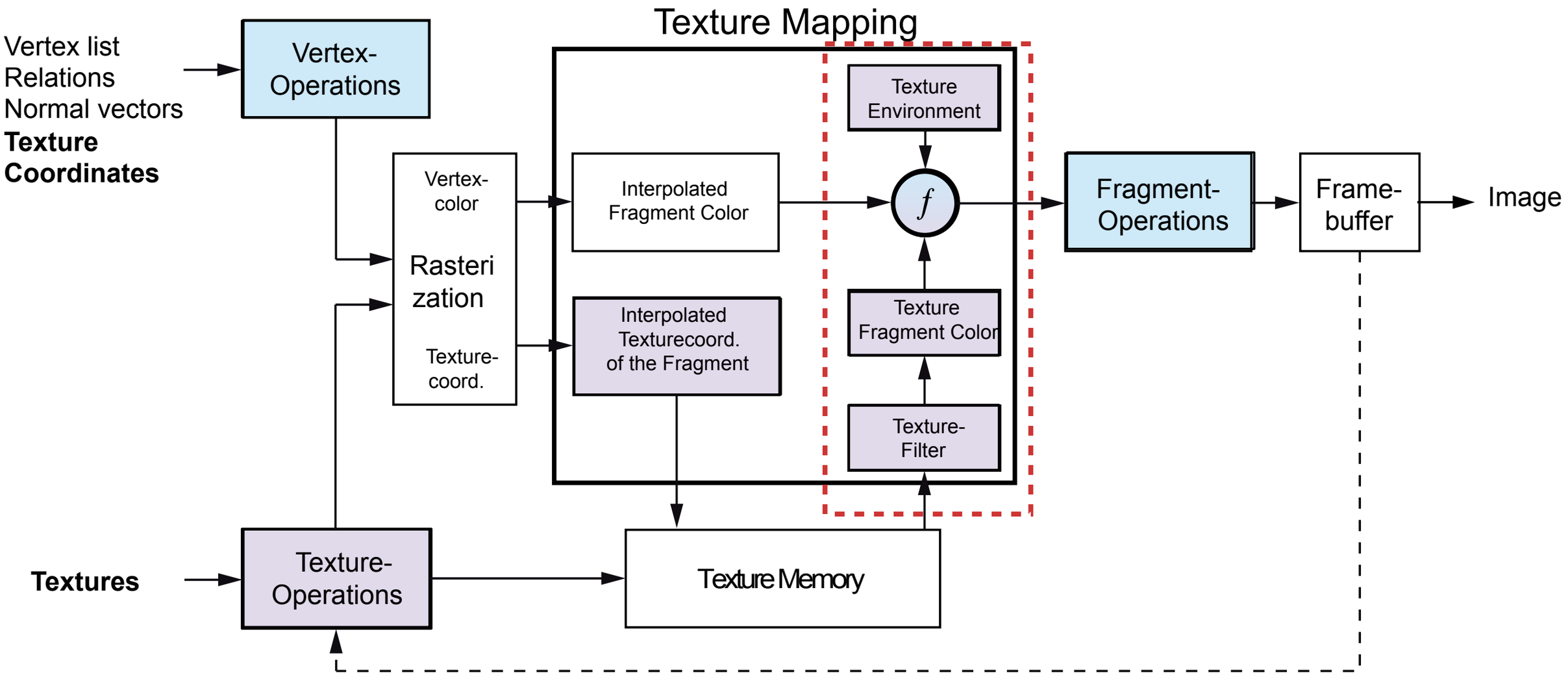




Texture coordinates rendered as colors

The OpenGL Rendering Architecture

We also know this architecture.

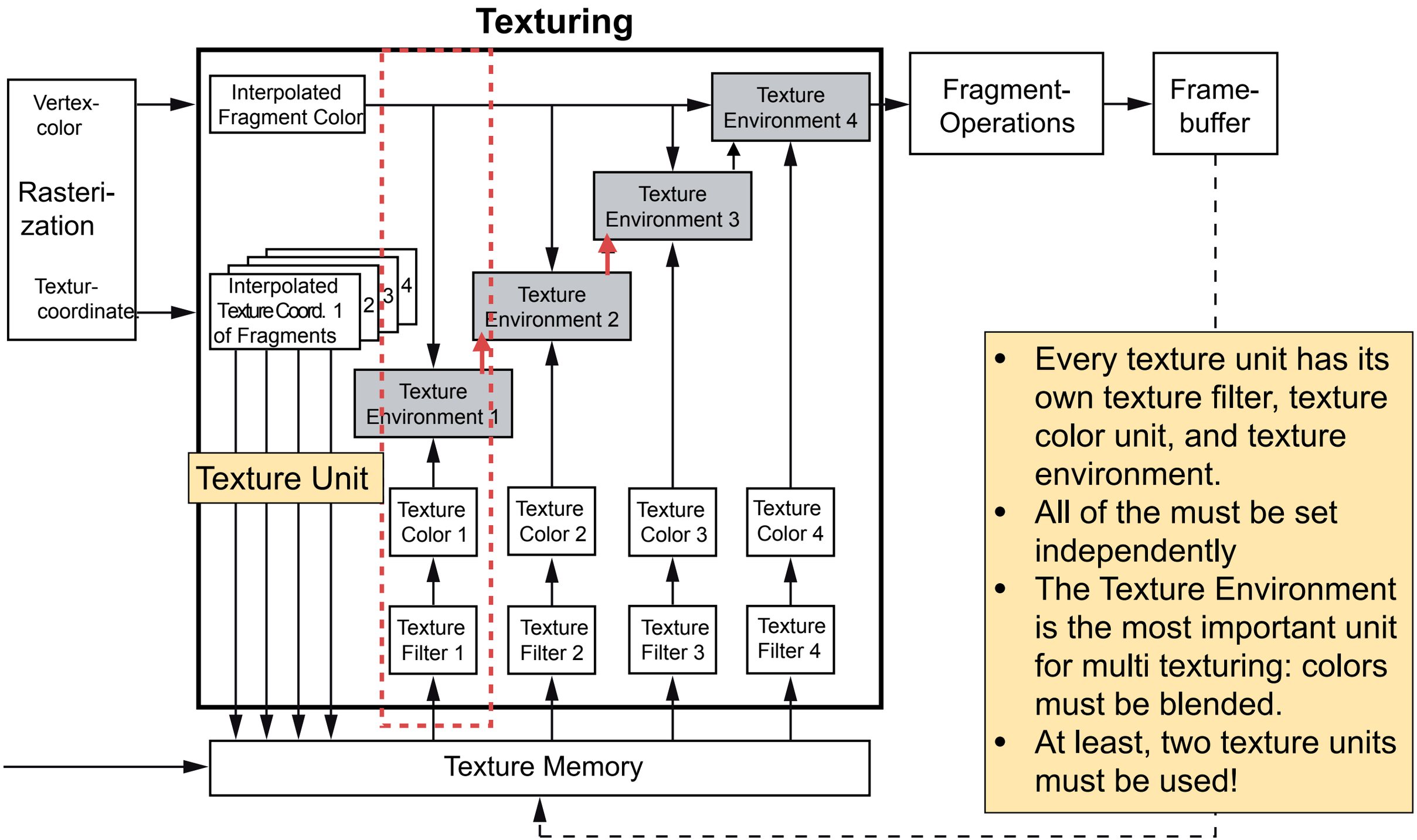


The OpenGL Rendering Pipeline shows the sequence of data processing inside the graphics card

Lighting Operations (last week class)

Texture Mapping

OpenGL Pipeline

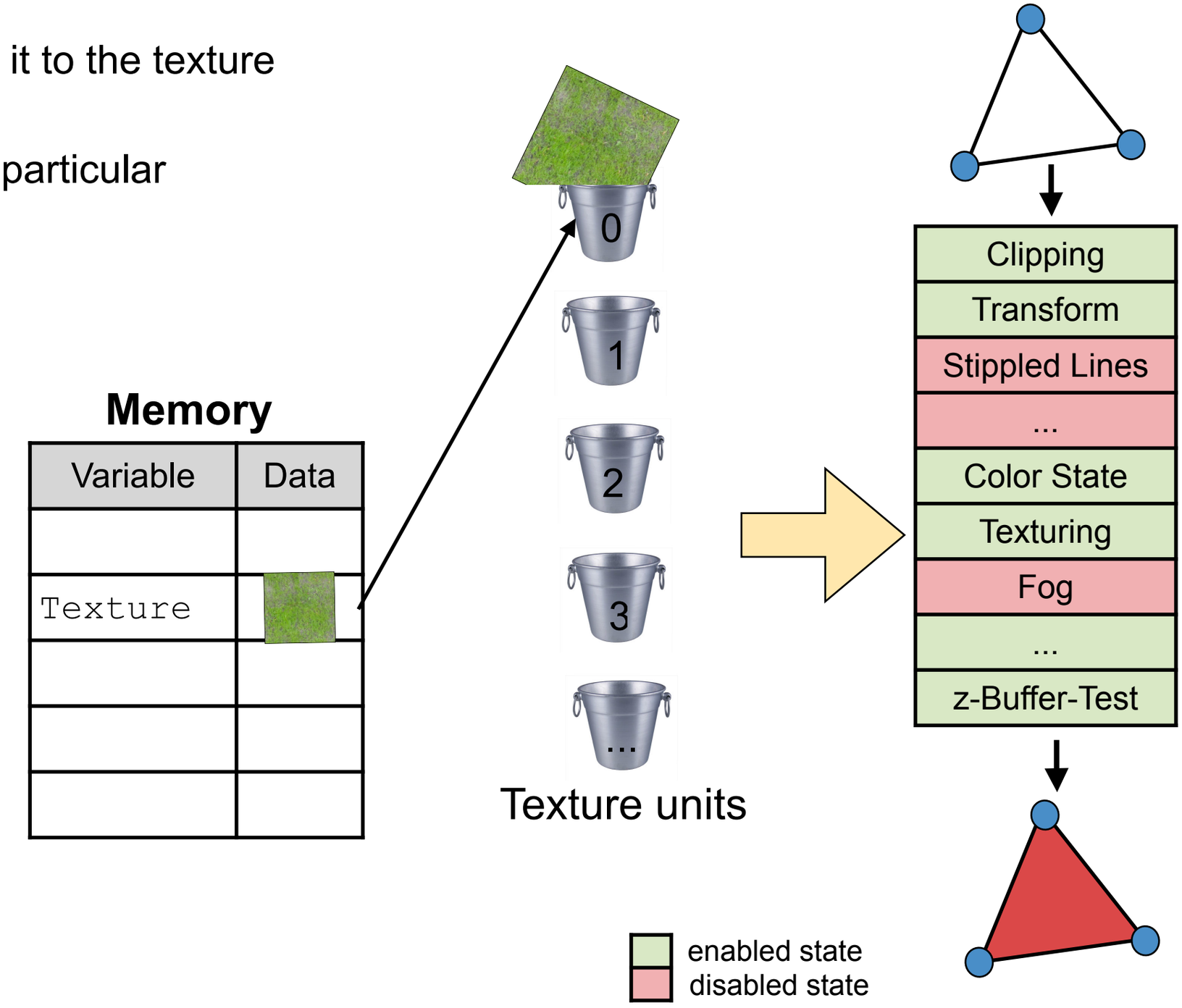


NVIDIA GTX Titan Z: 240 x 2 texture mapping units, 169 GTexel/s x2, 2880 x2 shading units
AMD Radeon R9 390 X2: 160 x2 texture mapping units, 160 GTexel/s x2, 2560 x2 shading units

Texturing Process

Two steps:

1. Defining textures and load it to the texture memory.
2. Associate a texture with a particular texture unit and apply it.



Assign a GPU name to your data

```
int main(int argc, const char * argv[])
{
        [....]

    while{

        [....]

    }
}
```

Program init

main loop

Texture definitions are invoked during the program init phase

```
glGenTextures(1, &_texture );
glBindTexture( GL_TEXTURE_2D, _texture );
```

CPU vs GPU program

ARLAB

CPU / host computer program

```
int main(int argc, const char * argv[])
{
.....
```

Texture declarations and all settings are made in the host program.

GPU shader program

```
static const string vs_string =
    "#version 410 core                                                 \n"
    "                                                                   \n"
    "uniform mat4 projectionMatrix;                                    \n"
    "uniform mat4 viewMatrix;                                           \n"
    "uniform mat4 modelMatrix;                                          \n"
    "in vec3 in_Position;                                               \n"
    "                                                                   \n"
    "in vec3 in_Color;                                                  \n"
    "out vec3 pass_Color;                                               \n"
```

The texturing process itself is carried out in the GPU program

glGenTextures

Generate the texture name

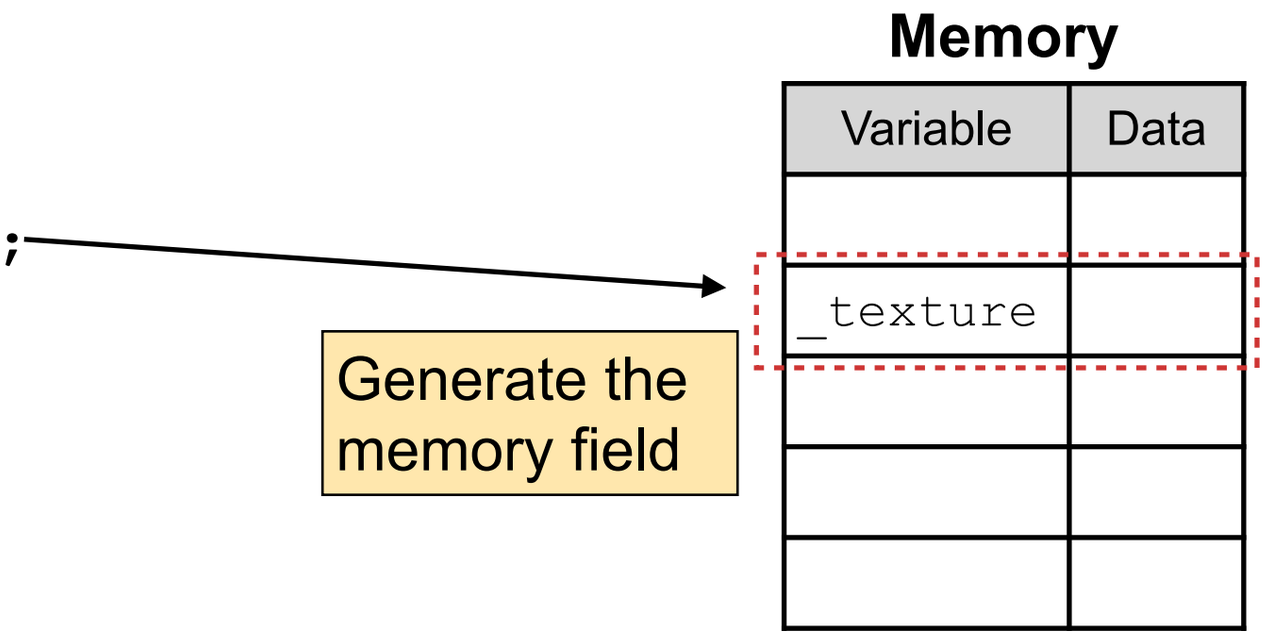
```
void glGenTextures( GLsizei n, GLuint * textures);
```

Parameters

- n - Specifies the number of texture names to be generated.
- textures - Specifies an array in which the generated texture names are stored.

Examples

```
GLuint _texture;
glGenTextures(1, &_texture );
```



Note, the texture is still not inside the memory and we have to link it with the memory.

glBindTexture

ARLAB

Bind a named texture to a texturing target

```
void glBindTexture(GLenum target, GLuint texture);
```

Parameters

- target - Specifies the target to which the texture is bound. Must be one of GL_TEXTURE_1D, ***GL_TEXTURE_2D***, GL_TEXTURE_3D, GL_TEXTURE_1D_ARRAY, GL_TEXTURE_2D_ARRAY, GL_TEXTURE_RECTANGLE, GL_TEXTURE_CUBE_MAP, GL_TEXTURE_CUBE_MAP_ARRAY, GL_TEXTURE_BUFFER, GL_TEXTURE_2D_MULTISAMPLE or GL_TEXTURE_2D_MULTISAMPLE_ARRAY.
- texture - Specifies the name of a texture.

- Consider the texture target as the goal that you want to achieve with this texture. However, you need to know your target. If you want to render the texture as part of the scene, your target is GL_TEXTURE_2D.
- The texture you bind is also the "active" texture. All further commands affect this texture.

Example

ARLAB

```
// Create a texture and load it to your graphics hardware. This
texture is automatically associated with texture 0 and the texture
variable "texture" / the active texture.
GLuint _texture;
glGenTextures(1, &_texture );

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

glTexParameterf

ARLAB

OpenGL function to setup a specific parameter

```
glTexParameterf ( GLenum target, GLenum pname, GLfloat param )
```

Parameters:

- 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.

This functions affect the current active texture. You have to make changes before you render the texture.

Example

ARLAB

```
// Create a texture and load it to your graphics hardware. This
texture is automatically associated with texture 0 and the texture
variable "texture" / the active texture.
GLuint _texture;
glGenTextures(1, &_texture );

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

// Change the parameters of your texture units.
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 );
```

- Must be the same target.
- This parameter changes affect the target GL_TEXTURE_2D
- It does NOT directly affect the variable _texture.

_texture

active

Create a texture

ARLAB

Create a texture and download the data to the memory.

```
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

```
// Change the parameters of your texture units.
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 );

// Create a texture and load it to your graphics hardware. This
texture is automatically associated with texture 0 and the texture
variable "texture" / the active texture.
if(channels == 3)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_BGR,
    GL_UNSIGNED_BYTE, data);
else if(channels == 4)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_BGRA,
    GL_UNSIGNED_BYTE, data);
```

Example

ARLAB

```
// Change the parameters of your texture units.
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 );

// Create a texture and load it to your graphics hardware. This
texture is automatically associated with texture 0 and the texture
variable "texture" / the active texture.
```

Amount of memory

Sequence of information

```
if(channels == 3)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_BGR,
    GL_UNSIGNED_BYTE, data);
else if(channels == 4)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_BGRA,
    GL_UNSIGNED_BYTE, data);
```

- Be aware of the number of channels, the number of color components (RGB =3 and RGBA = 4)
- Comes from your image file.

Example

ARLAB

```
// Change the parameters of your texture units.
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 );

// Create a texture and load it to your graphics hardware. This
texture is automatically associated with texture 0 and the texture
variable "texture" / the active texture.
```

```
if(channels == 3)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_BGR,
    GL_UNSIGNED_BYTE, data);
else if(channels == 4)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_BGRA,
    GL_UNSIGNED_BYTE, data);
```

- The texture data is associated with the target and not specifically with the variable _texture

Example

ARLAB

```
// Change the parameters of your texture units.
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 );

// Create a texture and load it to your graphics hardware. This
texture is automatically associated with texture 0 and the texture
variable "texture" / the active texture.
```

```
if(channels == 3)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_BGR,
    GL_UNSIGNED_BYTE, data);
else if(channels == 4)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_BGRA,
    GL_UNSIGNED_BYTE, data);
```

- The data type and the type of the data.

Assign a GPU name to your data

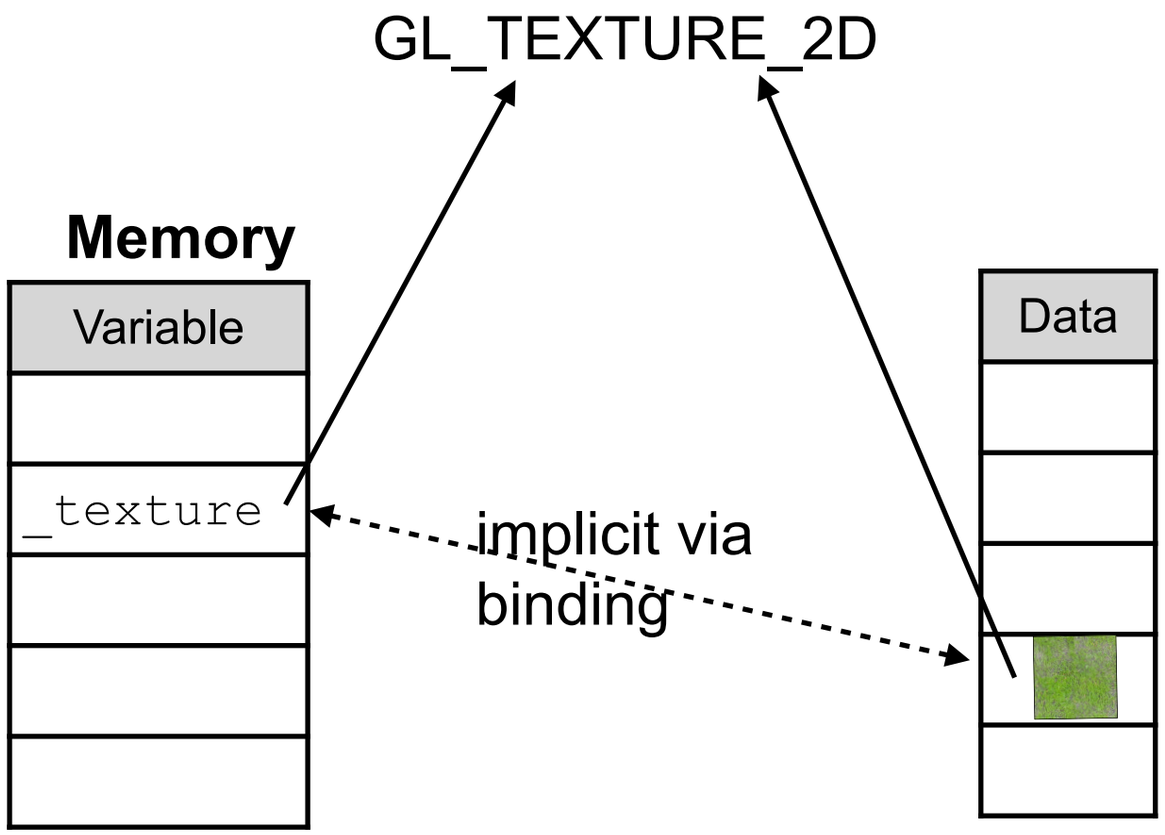
ARLAB

```
int main(int argc, const char * argv[])
{
        [....]
        glGenTextures(1, &_texture );
        glBindTexture( GL_TEXTURE_2D, _texture );

        glGenTextures(2, &_texture2 );
        glBindTexture( GL_TEXTURE_2D, _texture2 );

    while{

        [....]

    }
}
```

Program init

main loop

Relations



- We create a name that we can use a logical handler in our program to access the texture and to apply it.
- The _texture variable is tied to a target GL_TEXTURE_2D
- We download the texture data to our memory.
- The data is tied to a target GL_TEXTURE_2D

Create a uniform variable

Create a uniform variable

ARLAB

CPU / host computer program

```
int main(int argc, const char * argv[])
{
.....
```

Graphic Memory

Variable	Data
_texture	Data

We have to create a uniform variable to access the texture in our GPU program.

GPU shader program

```
static const string vs_string =
    "#version 410 core
    "
    "uniform mat4 projectionMatrix;
    "uniform mat4 viewMatrix;
    "uniform mat4 modelMatrix;
    "in vec3 in_Position;
    "
    "in vec3 in_Color;
    "out vec3 pass_Color;
```

```
\n"
 \n"
\n"
 \n"
 \n"
 \n"
 \n"
 \n"
 \n"
```

Create a uniform variable

ARLAB

These are the steps we have to follow:

1. Enable the glsl shader program or make sure that it is already enabled.

```
// enable the program
glUseProgram(program);
```

2. Query the location of the uniform variable and store the index

```
int idx = glGetUniformLocation(program, "tex" );
```

```
1  #version 410 core
2
3  uniform sampler2D tex; //this is the texture
4
5  in vec2 pass_TexCoord; //this is the texture coord
6  in vec4 pass_Color;
7  out vec4 color;
8
9  uniform int texture_blend;
10
11 void main(void)
12 {
```

Snippet of the GLSL shader program

glGetUniformLocation

ARLAB

Returns the location of a uniform variable

```
GLint glGetUniformLocation( GLuint program, const GLchar *name);
```

Parameters

- program - Specifies the program object to be queried.
- name - Points to a null terminated string containing the name of the uniform variable whose location is to be queried.

Note,

- the uniform variable must share the same name.
- the glsl program must exist, means, it must have been already compiled and linked

Create a uniform variable

ARLAB

These are the steps we have to follow:

1. Enable the glsl shader program or make sure that it is already enabled.

```
// enable the program
glUseProgram(program);
```

2. Query the location of the uniform variable and store the index

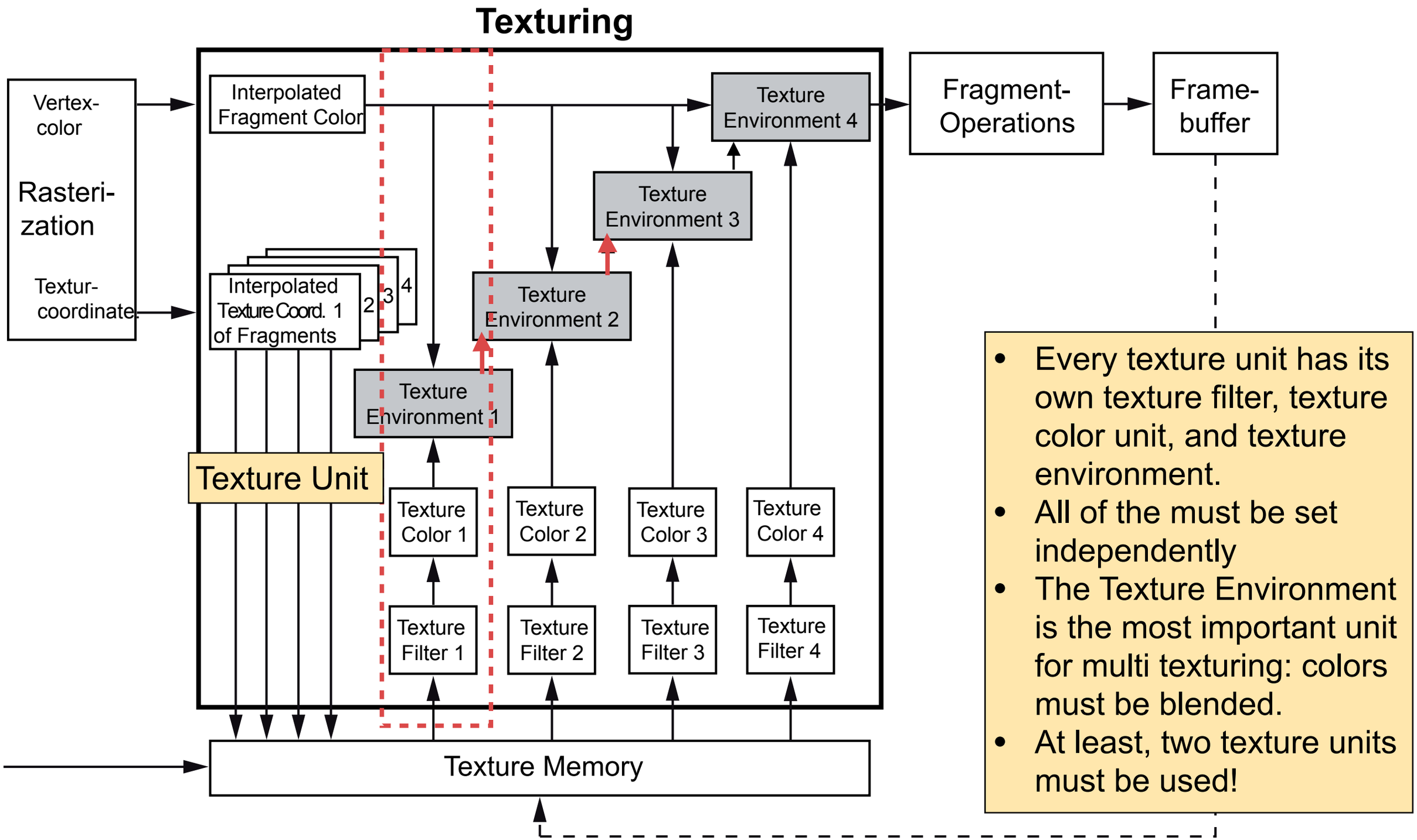
```
int idx = glGetUniformLocation(program, "tex" );
```

3. Activate a texture unit

```
glActiveTexture(GL_TEXTURE0);
```

- This activates texture unit 0
- Make sure you know how many you have.
  Desktop computer: 8, smart phone: 1 or 2

OpenGL Pipeline



NVIDIA GTX Titan Z: 240 x 2 texture mapping units, 169 GTexel/s x2, 2880 x2 shading units
AMD Radeon R9 390 X2: 160 x2 texture mapping units, 160 GTexel/s x2, 2560 x2 shading units

glActiveTexture

ARLAB

Select a texture unit

```
void glActiveTexture(GLenum texture);
```

Parameters:

texture - Specifies which texture unit to make active. The number of texture units is *implementation dependent*, but must be at least 8.

Implementation dependent: it depends on your hardware!

Create a uniform variable

ARLAB

2. Query the location of the uniform variable and store the index

```
int idx = glGetUniformLocation(program, "tex" );
```

3. Activate a texture unit

```
glActiveTexture(GL_TEXTURE0);
```

4. Bind a texture - this texture is now associated with the active texture unit. We bind the texture into the active hardware unit.

```
glBindTexture(GL_TEXTURE_2D, _texture);
```

Create a uniform variable

ARLAB

2. Query the location of the uniform variable and store the index

```
int idx = glGetUniformLocation(program, "tex" );
```

3. Activate a texture unit

```
glActiveTexture(GL_TEXTURE0);
```

4. Bind a texture - this texture is now associated with the active texture unit. We bind the texture into the active hardware unit.

```
glBindTexture(GL_TEXTURE_2D, _texture);
```

5. Create a uniform variable:

```
glUniform1i(idx, 0);
```

Create a uniform variable

2. Query the location of the uniform variable and store the index

```
int idx = glGetUniformLocation(program, "tex" );
```

3. Activ

```
glAc                                    );
```

As before, we always bind a uniform value by using the location of this value inside the shader program.

4. Bind a texture - this texture is now associated with the active texture unit.
   We bind the texture into the active hardware unit.

```
glBindTexture(GL_TEXTURE_2D, _texture);
```

5. Create a uniform variable:

```
glUniform1i(idx, 0);
```

Create a uniform variable

ARLAB

2. Query the location of the uniform variable and store the index

```
int idx = glGetUniformLocation(program, "tex" );
```

3. Activate a texture unit

```
glActiveTexture(GL_TEXTURE0);
```

4. Bind a texture - this texture is now associated with the active texture unit.
   We bind the texture into the active hardware unit.

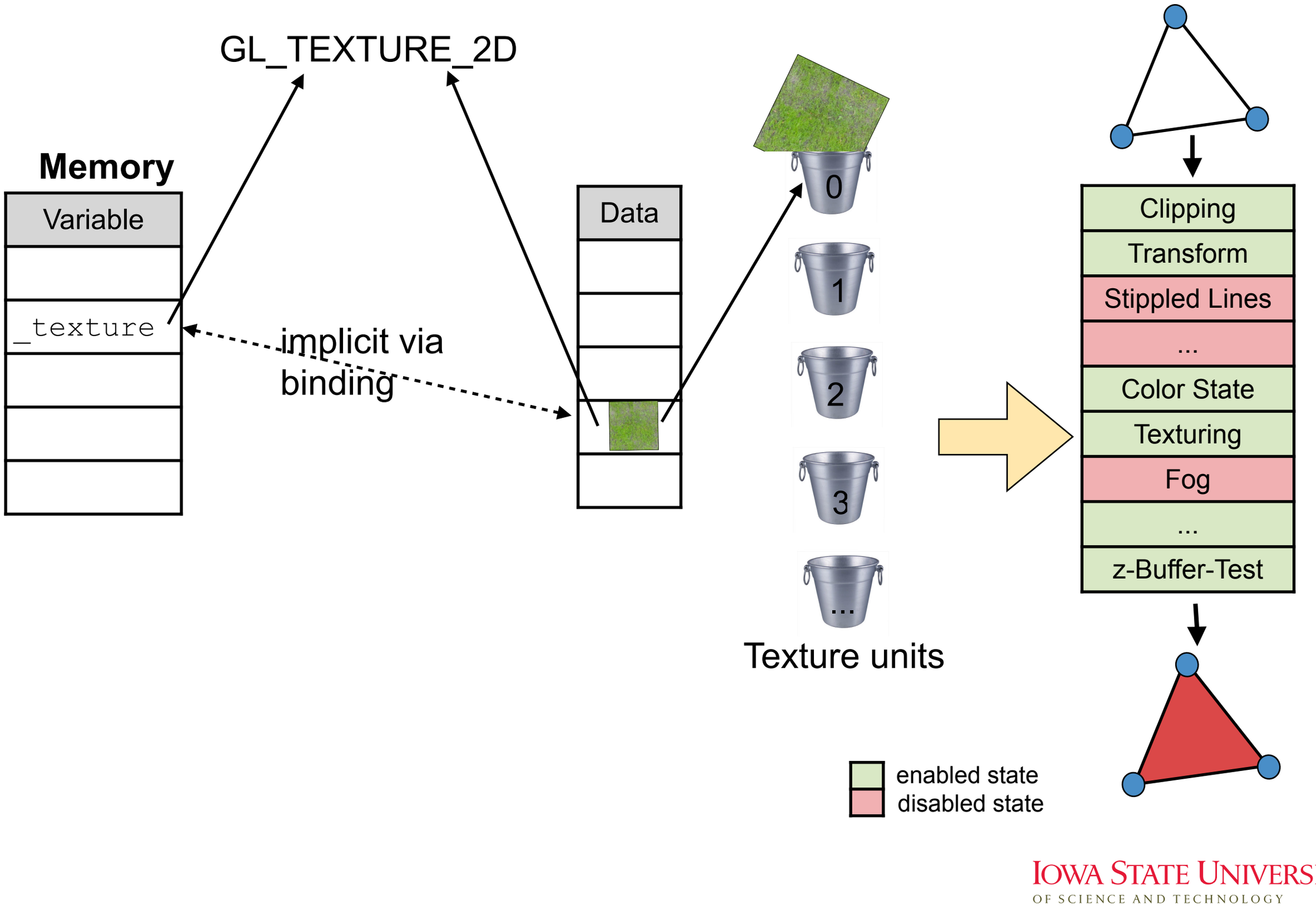
```
glBindTexture(GL_TEXTURE_2D
```

5. Create a uniform variable:

```
glUniform1i(idx, 0);
```

But instead of sending the texture to the glsl shader, we send the number of the active texture unit that should process this texture.
In this example, this is 0 because we use texture unit 0 to process the texture.

Relations


GL_TEXTURE_2D
Memory
Variable
texture
_texture
implicit via
binding
Data
0
1
2
3
...
Texture units
Clipping
Transform
Stippled Lines
...
Color State
Texturing
Fog
...
z-Buffer-Test
enabled state
disabled state

Max Number of Texture Units

The function glGet returns a selected value

```
void glGetIntegerv(GLenum pname, GLint * params);
void glGetFloat(GLenum pname, GLfloat * params);
```

[some more exist, see specification]

Parameter:

- pname: the name of the parameter
- params: a variable in which OpenGL returns the value you asked for,
  - GL_MAX_TEXTURE_UNITS: return the number of max. texture units

Make sure that you know the correct return type. Look into the OpenGL specifications.

To return the maximum number of texture units of your graphics card:

Example:

```
GLint iUnits;
glGetIntegerv(GL_MAX_TEXTURE_UNITS, &iUnits);
```

Texture Units vs Texture Bind

- Texture units (glActiveTexture) and texture bind operation (glBindTexture) are different functions.
- glActiveTexture determines which texture unit (an electric circuit on your graphics card) shall process the texture.
- glBindTexture logically combines a texture with a primitive.
- All textures of one application can processed by one texture unit.
- Use multiple texture units only for multi-texturing

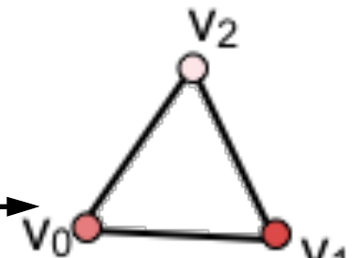
ARLAB

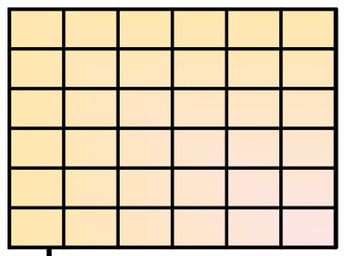
Map the texture to the primitive

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Texture Mapping: get a pixel color

Display List of a 3D model

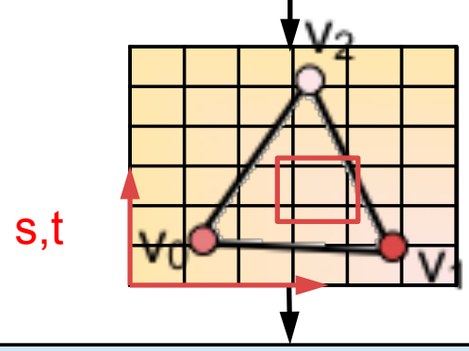




Texture

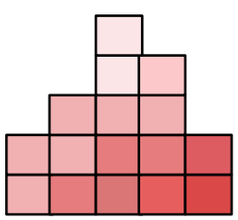
Rasterization

- 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

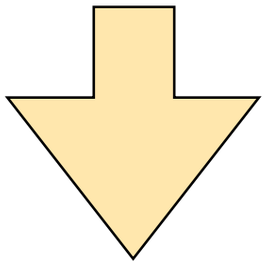
Fragment Shader Program

ARLAB

This is fragment shader program simple_texture.fs

```
#version 410 core

uniform sampler2D tex; //this is the texture

in vec2 pass_TexCoord; //this is the texture coord
in vec4 pass_Color;
out vec4 color;

uniform int texture_blend;

void main(void)
{
    // This function finds the color component for each texture coordinate.
    vec4 tex_color =  texture(tex, pass_TexCoord);

    // This mixes the background color with the texture color.
    // The GLSL shader code replaces the former envrionment. It is now up to us
    // to figure out how we like to blend a texture with the background color.
    if(texture_blend == 0)
    {
        color = pass_Color + tex_color;
    }
    else if(texture_blend == 1)
    {
        color = pass_Color * tex_color;
    }
    else if(texture_blend == 3)
    {
        color = (1-pass_Color.w)*pass_Color + tex_color;
    }
    else
    {
        color = pass_Color + tex_color;
    }

}
```

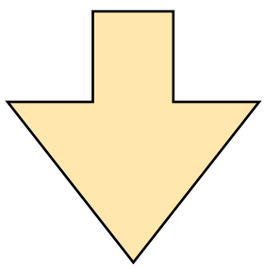
Fragment Shader Program



```
in vec2 pass_TexCoord;
in vec4 pass_Color;
```

```
#version 410 core

uniform sampler2D tex; //this is the texture

in vec2 pass_TexCoord; //this is the texture coord
in vec4 pass_Color;
out vec4 color;
```

```
void main(void)
{
    // This function finds the color component for each texture coordinate.
    vec4 tex_color =  texture(tex, pass_TexCoord);

    // This mixes the background color with the texture color.
    // The GLSL shader code replaces the former envrionment. It is now up to us
    // to figure out how we like to blend a texture with the background color.
    color = pass_Color + tex_color;
}
```



```
out vec4 color;
```

Fragment Shader Program

ARLAB

```
#version 410 core

uniform sampler2D tex; //this is the texture

in vec2 pass_TexCoord; //this is the texture coord
in vec4 pass_Color;
out vec4 color;
```

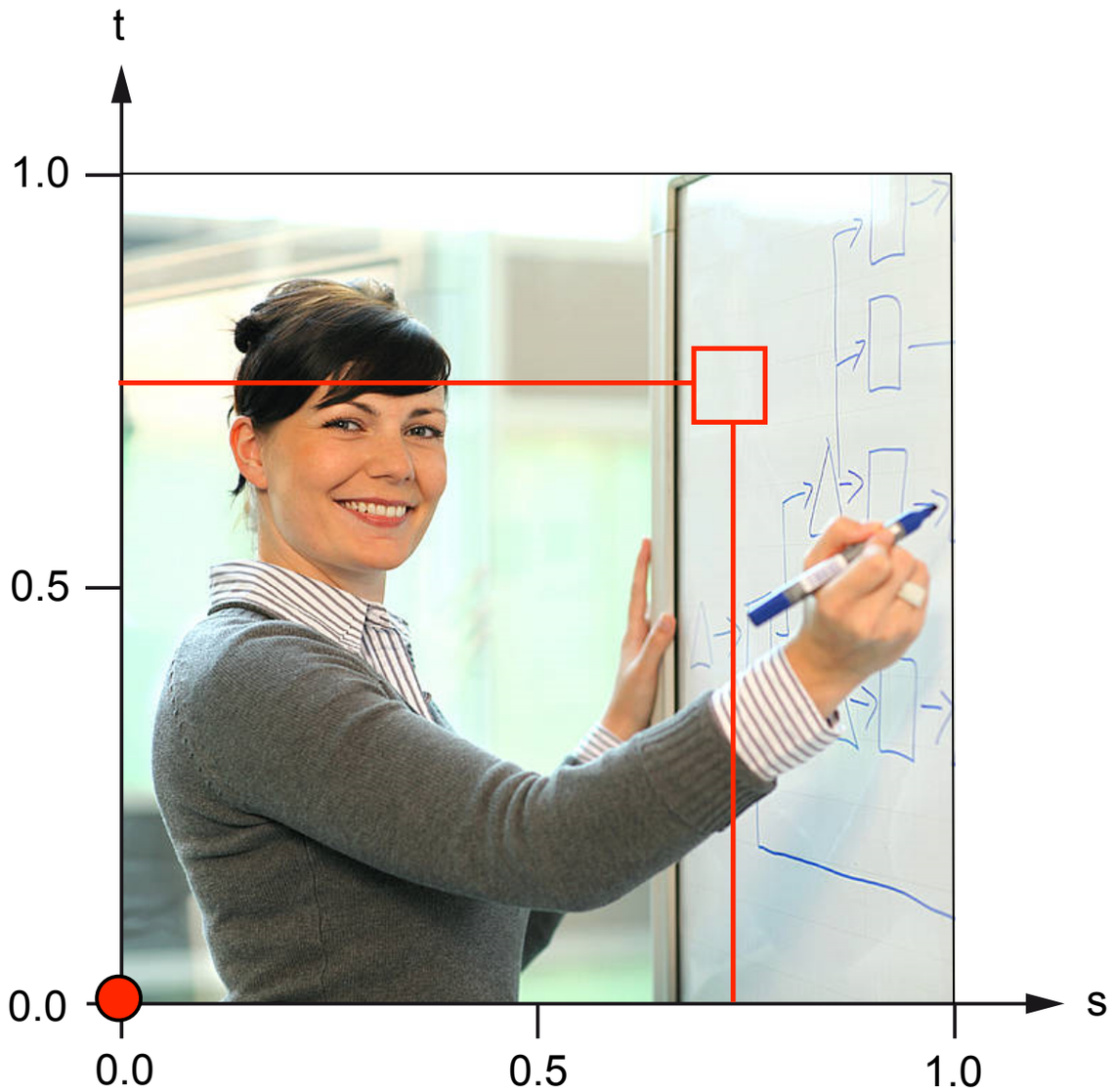
This is the texture

```
void main(void)
{
    // This function finds the color component for each texture coordinate.
    vec4 tex_color =  texture(tex, pass_TexCoord);

    // This mixes the background color with the texture color.
    // The GLSL shader code replaces the former envrionment. It is now up to us
    // to figure out how we like to blend a texture with the background color.
    color = pass_Color + tex_color;
}
```

sampler2D

A sampler2D is a GLSL variable type of the variable set sampler. We can use it to describe a texture.

```
uniform sampler2D tex; //this is the texture
```



A sampler must be a uniform variable since it can be only passed to the glsl fragment shader program via an active texture unit.

Fragment Shader Program

ARLAB

```
#version 410 core

uniform sampler2D tex; //this is the texture

in vec2 pass_TexCoord; //this is the texture coord
in vec4 pass_Color;
out vec4 color;

void main(void)
{
    // This function finds the color component for each texture coordinate.
    vec4 tex_color =  texture(tex, pass_TexCoord);

    // This mixes the background color with the texture color.
    // The GLSL shader code replaces the former envrionment. It is now up to us
    // to figure out how we like to blend a texture with the background color.
    color = pass_Color + tex_color;
}
```

This is the texture

Fragment Shader Program

ARLAB

```
#version 410 core

uniform sampler2D tex; //this is the texture

in vec2 pass_TexCoord; //this is the texture coord
in vec4 pass_Color;
out vec4 color;
```

```
void main(void)
{
    // This function finds the color component for each texture coordinate.
    vec4 tex_color =  texture(tex, pass_TexCoord);

    // This mixes the background color with the texture color.
    // The GLSL shader code replaces the former envrionment. It is now up to us
    // to figure out how we like to blend a texture with the background color.
    color = pass_Color + tex_color;
}
```

This function adds the texture color to our primitive color

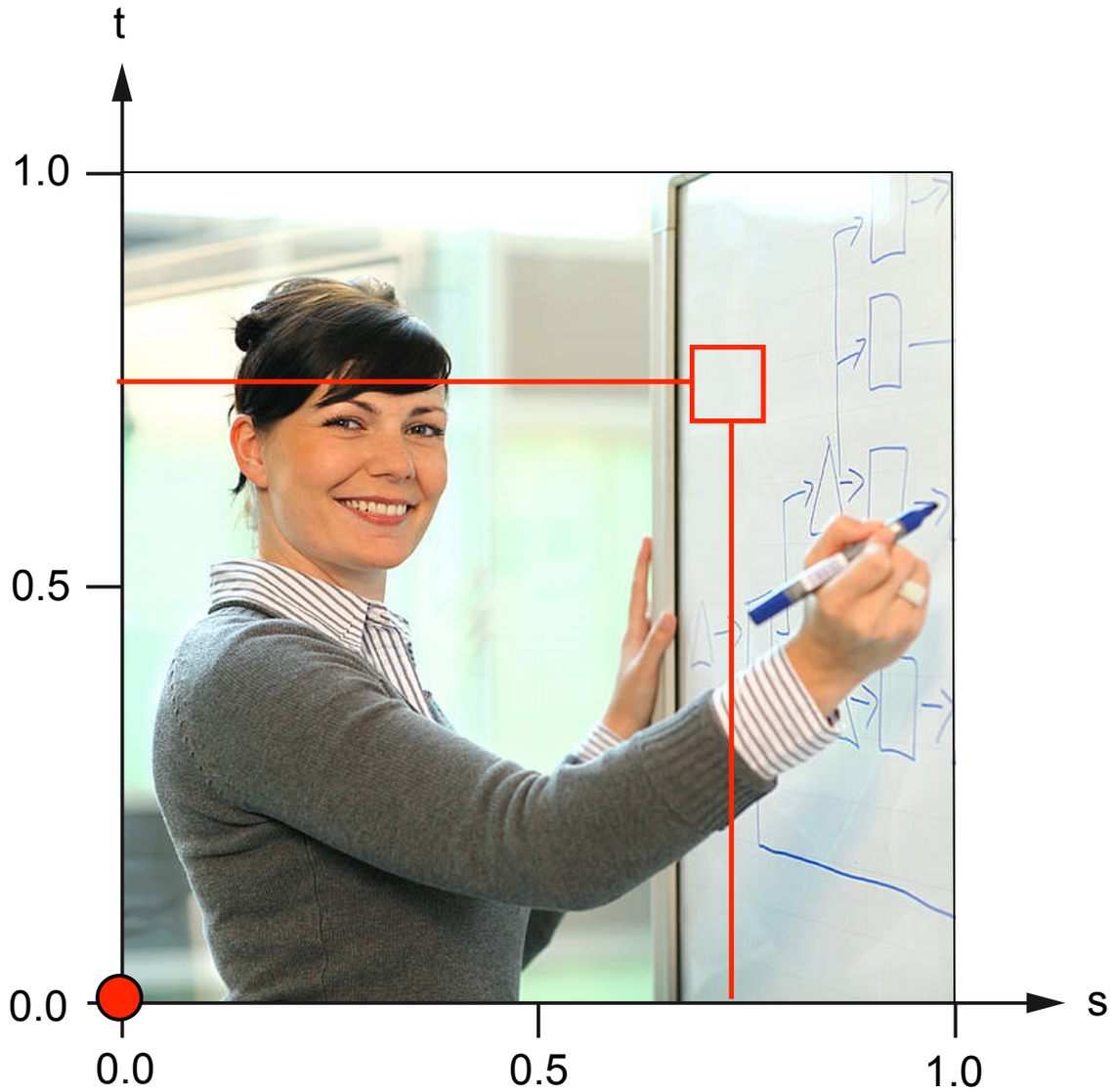
texture

Retrieves texels from a texture

```
gvec4 texture(gsampler2D sampler, vec2 P, [float bias]);
```

Parameters

- sampler - Specifies the sampler to which the texture from which texels will be retrieved is bound.
- P - Specifies the texture coordinates at which texture will be sampled.
- bias- Specifies an optional bias to be applied during level-of-detail computation.



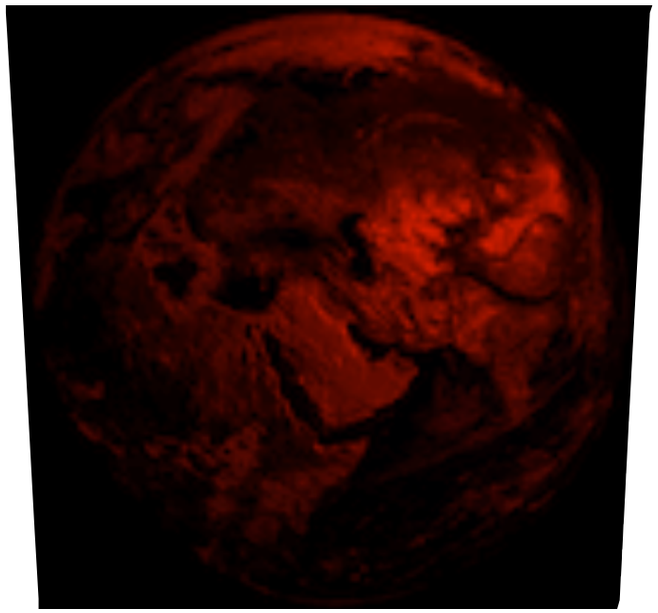
Blend Modes

Primitive

*

Texture (GL_RGB)

=



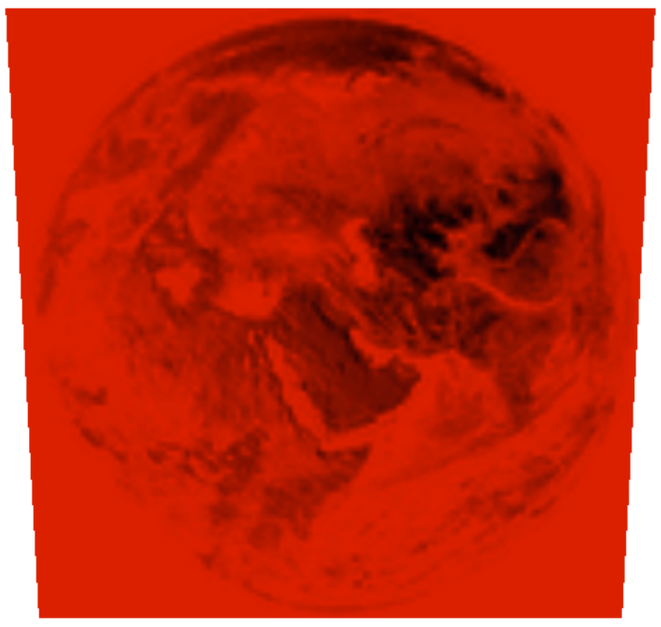
Result

Primitive

+

Texture (GL_RGB)

=



Result

Fragment Shader Program

ARLAB

```
#version 410 core

uniform sampler2D tex; //this is the texture

in vec2 pass_TexCoord; //this is the texture coord
in vec4 pass_Color;
out vec4 color;
```

```
void main(void)
{
    // This function finds the color component for each texture coordinate.
    vec4 tex_color =  texture(tex, pass_TexCoord);

    // This mixes the background color with the texture color.
    // The GLSL shader code replaces the former environment. It is now up to us
    // to figure out how we like to blend a texture with the background color.
    color = pass_Color + tex_color;
}
```

This is the blend mode, it is up to us now.

Fragment Shader Program

ARLAB

```
// This function finds the color component for each texture coordinate.
vec4 tex_color =  texture(tex, pass_TexCoord);

// This mixes the background color with the texture color.
// The GLSL shader code replaces the former envrionment. It is now up to us
// to figure out how we like to blend a texture with the background color.
if(texture_blend == 0)
{
    color = pass_Color + tex_color;
}
else if(texture_blend == 1)
{
    color = pass_Color * tex_color;
}
else if(texture_blend == 3)
{
    color = (1-pass_Color.w)*pass_Color + tex_color;
}
else
{
     color = pass_Color + tex_color;
}

}
```

The example program provides different blend modes.

This is fragment shader program simple_texture.fs

Vertex Shader Program

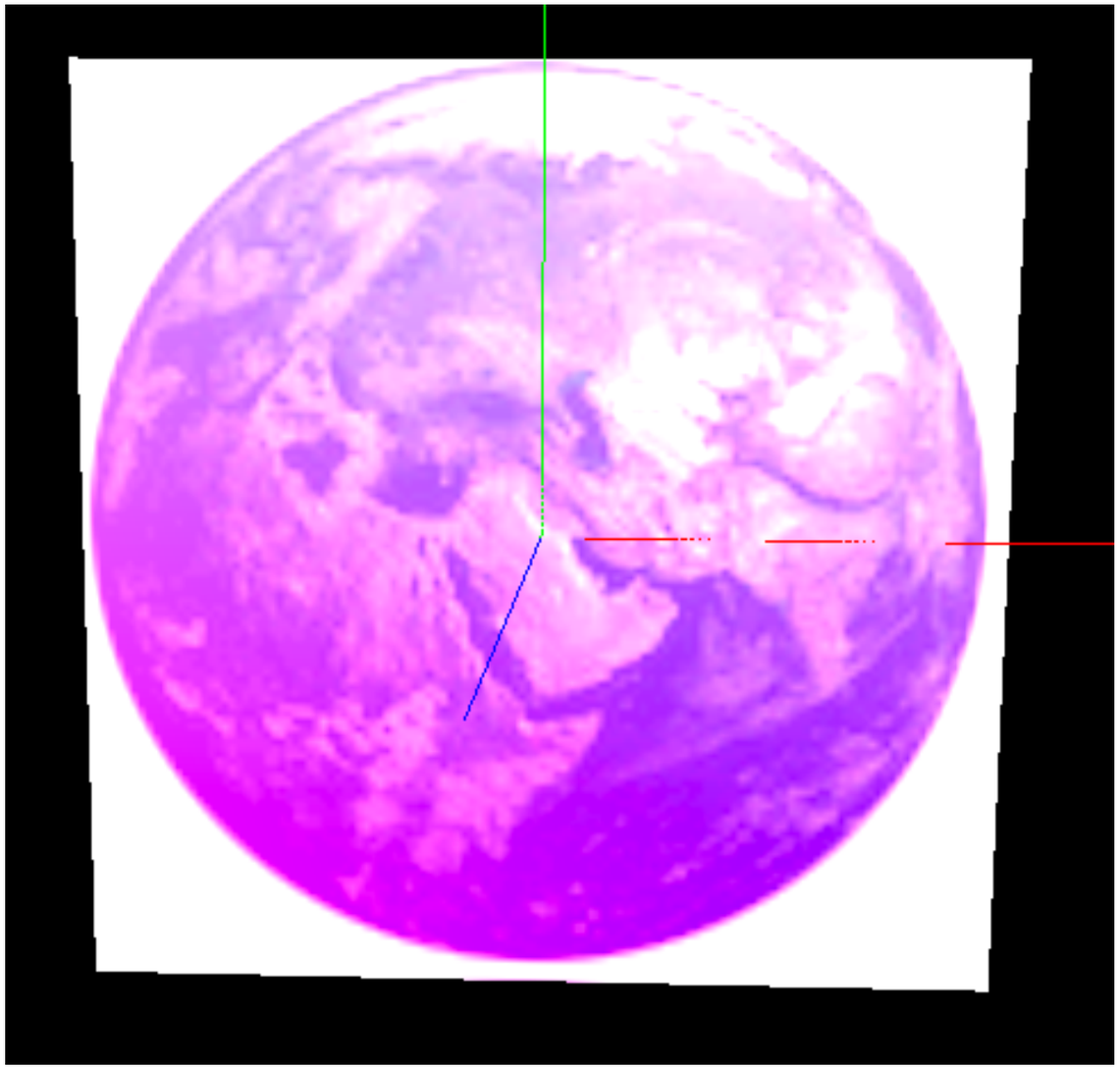
ARLAB

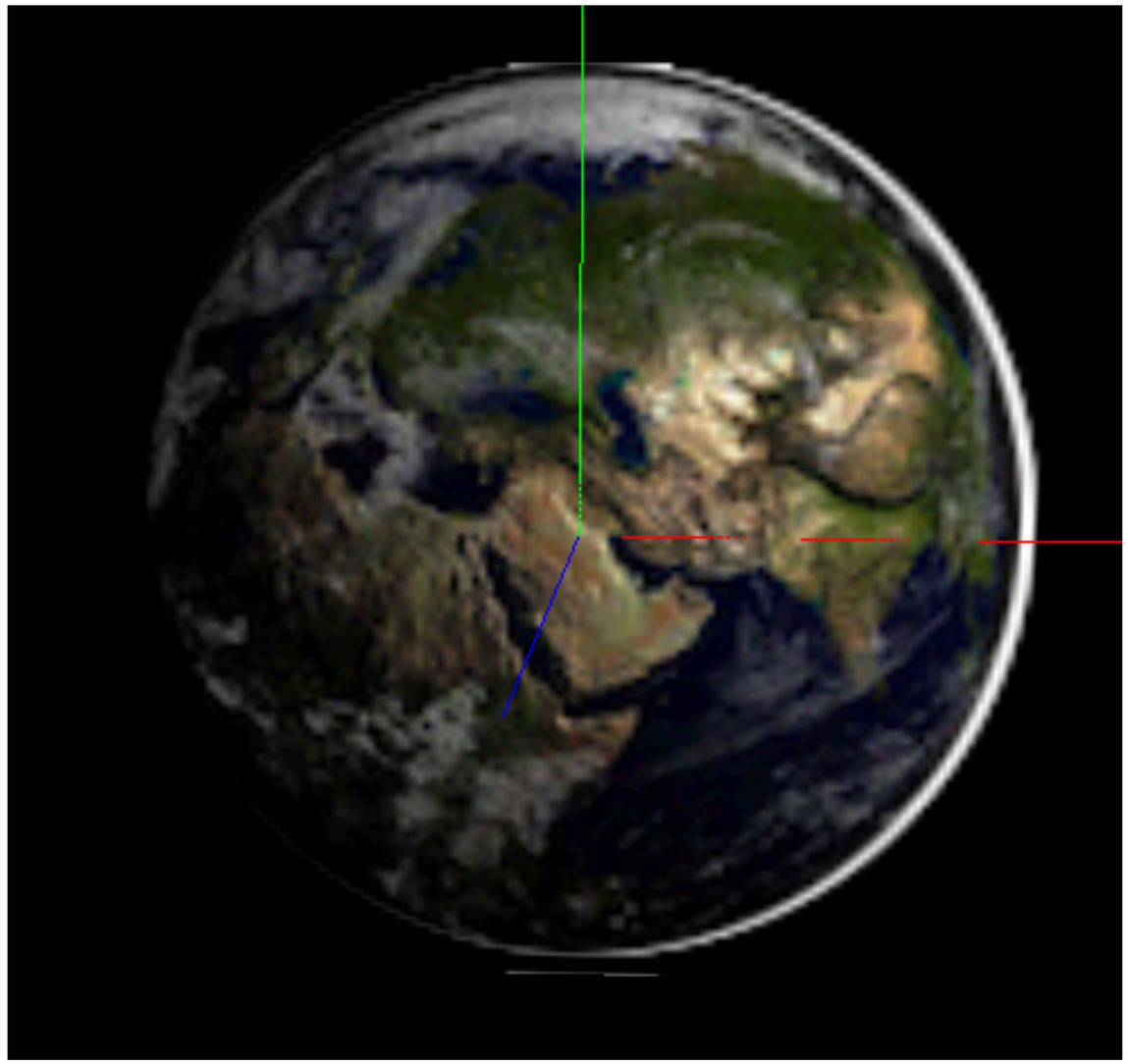
```
// The vertex buffer input
in vec2 in_TexCoord;
in vec3 in_Position;
in vec3 in_Normal;

// The output color
out vec4 pass_Color;
out vec2 pass_TexCoord;

    [...]

void main(void)
{
    [...]

    // Passes the texture coordinates to the next pipeline processes.
    pass_TexCoord = in_TexCoord;
}
```

Read the texture coordinates as incoming varying variable and pass the coordinates to the fragment shader program.

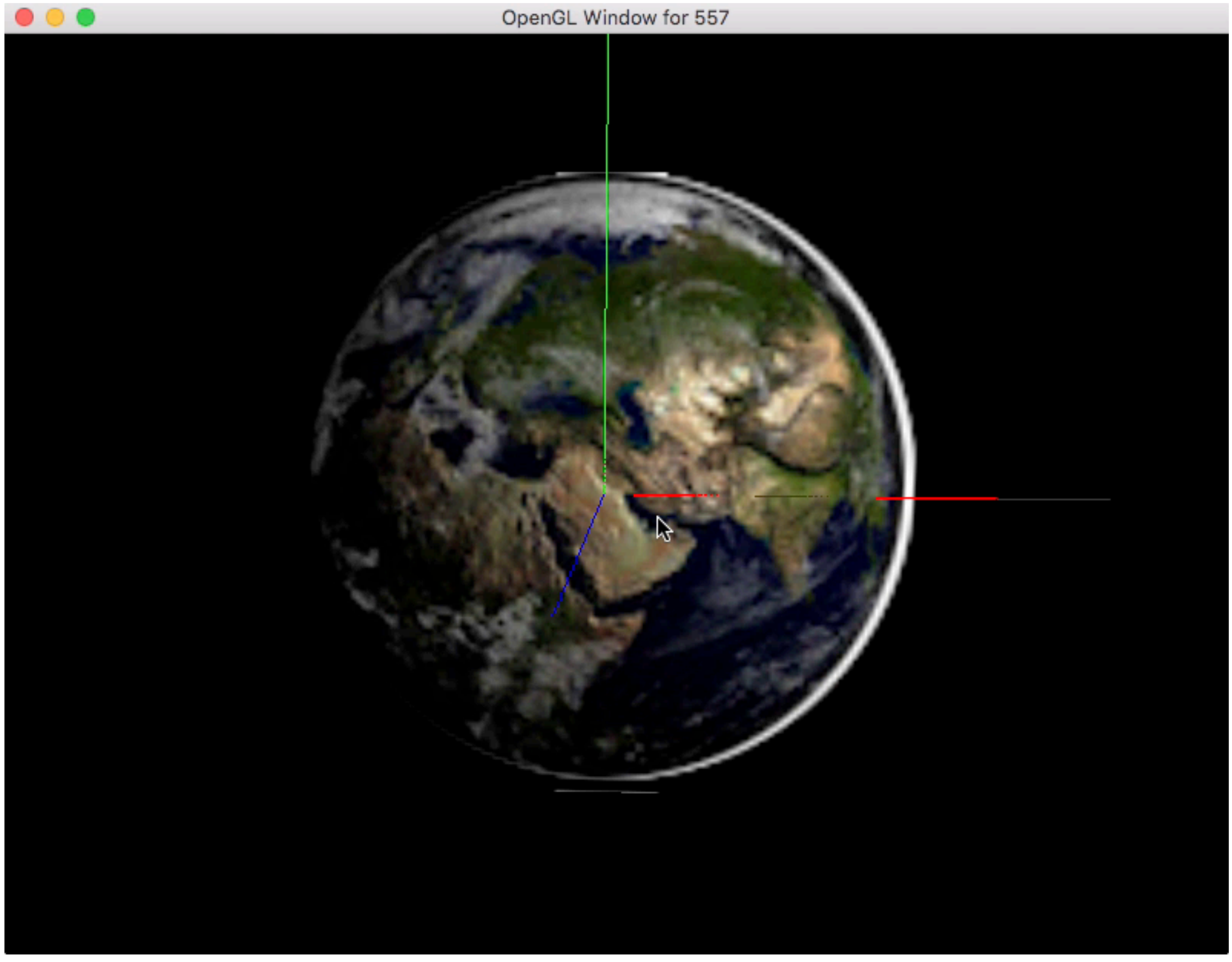
Example Code

ARLAB





Video



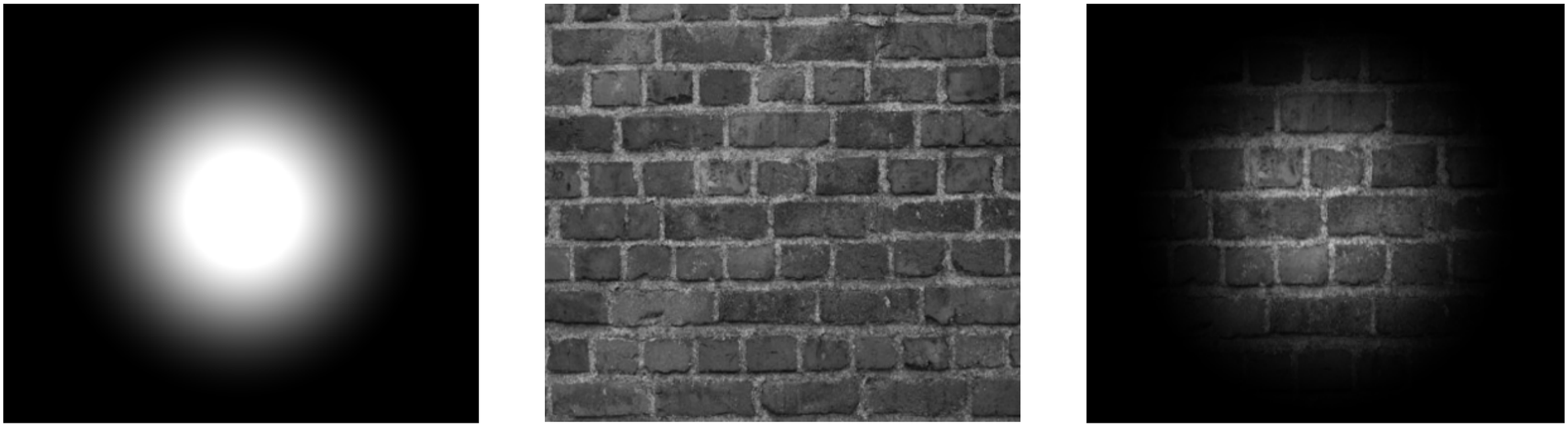
VRAC | HCI

Multi-texturing

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Multi -Texturing

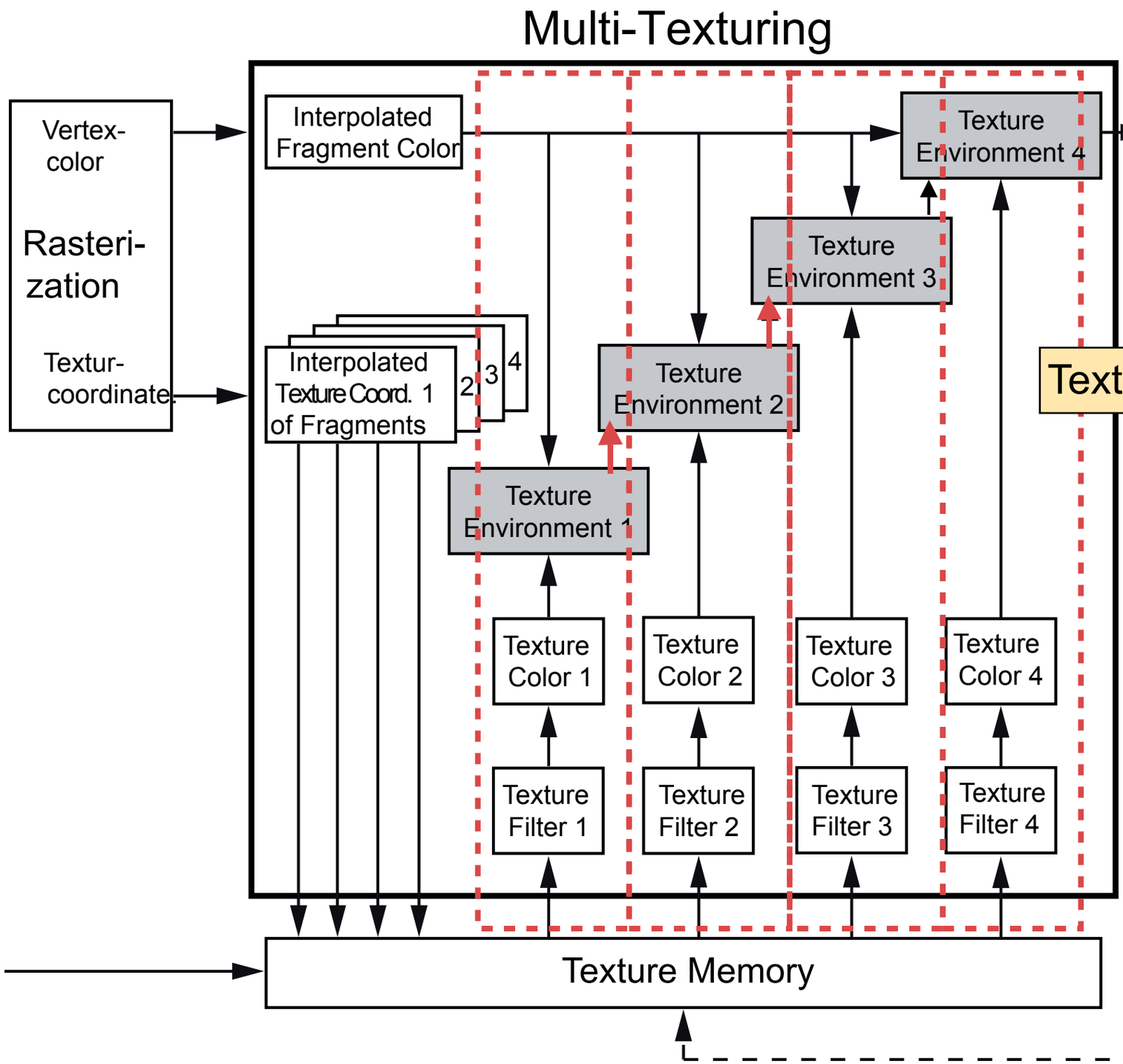
Multi-texturing combines more than one image (texture source) one one primitive.



(a) (b) (c)

The textures a) and b) are mapped to the primitive c). It appears as an illuminated brick wall.

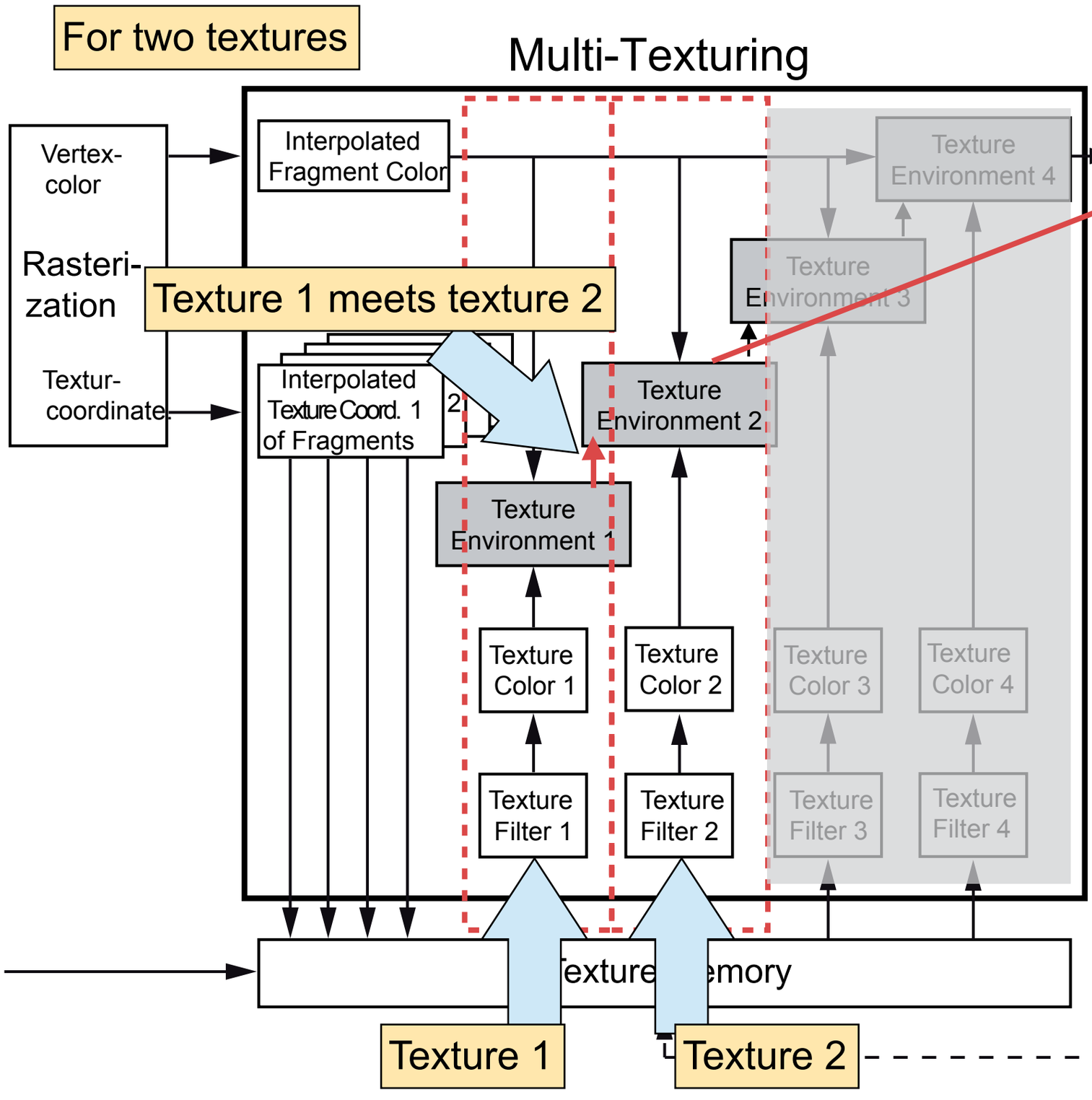
OpenGL Pipeline



- Every texture unit has its own texture filter, texture color unit, and texture environment.
- All of the must be set independently
- The Texture Environment is the most important unit for multi texturing: colors must be blended.
- At least, two texture units must be used!

NVIDIA GTX 780: 192 texture mapping units
AMD Radeon HD 7990 Graphics: 256 texture mapping units

OpenGL Pipeline



- Every texture unit has its own texture filter, texture color unit, and texture environment.
- All of the must be set independently
- The Texture Environment is the most important unit for multi texturing: colors must be blended.
- At least, two texture units must be used!

Process

Follow these steps to enable multi-texturing

1. Load both textures into the main memory.
2. Enable texture unit 0
3. Create the first texture
    1. Create memory
    2. Bind texture to memory
    3. Set parameters
    4. Load texture
4. Enable texture unit 1
5. Create the second texture
    1. Create memory
    2. Bind texture to memory
    3. Set parameters

    Make sure that the texture environment blends the color information of both textures

    4. Load texture

Preparation

---

6. Assign multi texture coordinates to a primitive surface.

Runtime

The new part: two textures has to be created in two texture units. Therefore, the texture units must be individually enabled.

Texture Units

VRAC | HCI

You can change the current texture unit by calling glActiveTexture with the texture unit identifier as the argument.

`void glActiveTexture( GLenum texture)`

Parameters:

- texture: Specifies which texture unit to make active, GL_TEXTURE0..... GL_TEXTURE8

- Each texture unit has its own texture environment that determines how fragments are combined with the previous texture unit.
- By default, the first texture unit is the active texture unit.
- All texture commands, with the exception of glTexCoord, affect the currently active texture unit.
- During rendering, the texture is applied from all enabled texture units.

```
glActiveTexture(GL_TEXTURE1);
glEnable(GL_TEXTURE_2D);
```

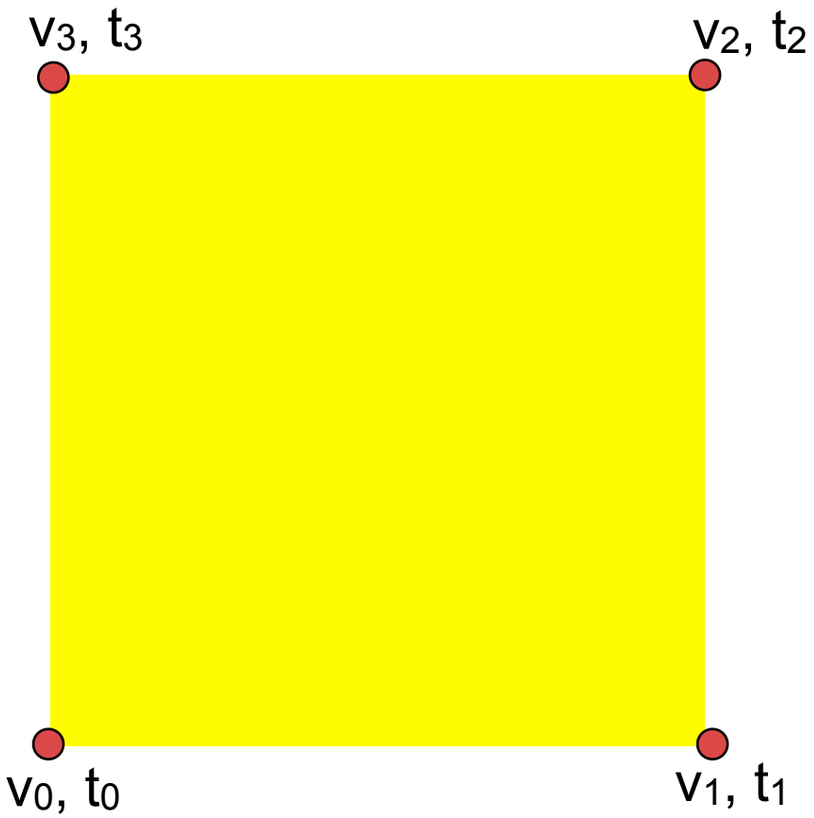
Enables texturing with texture unit 1

```
[.........]

glActiveTexture(GL_TEXTURE0);
glEnable(GL_TEXTURE_2D);
```

Enables texturing with texture unit 0

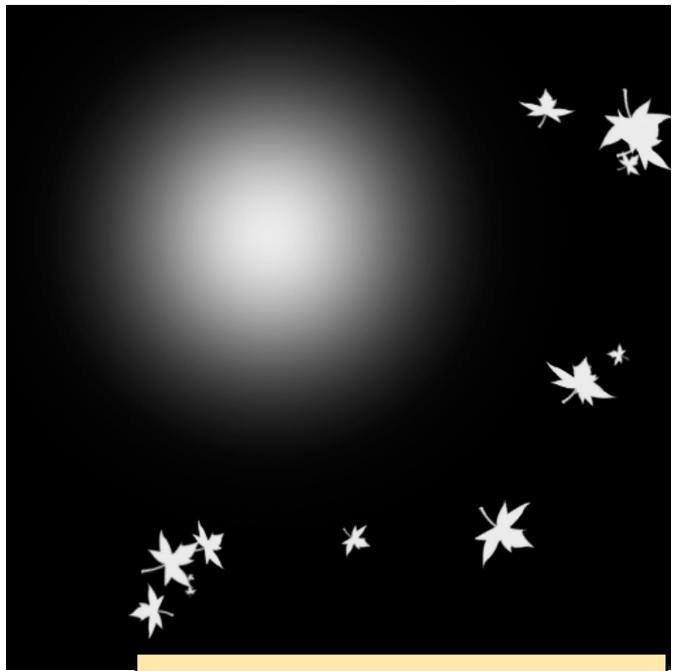
Example

A brick texture and a light texture should be mapped onto a yellow quad.



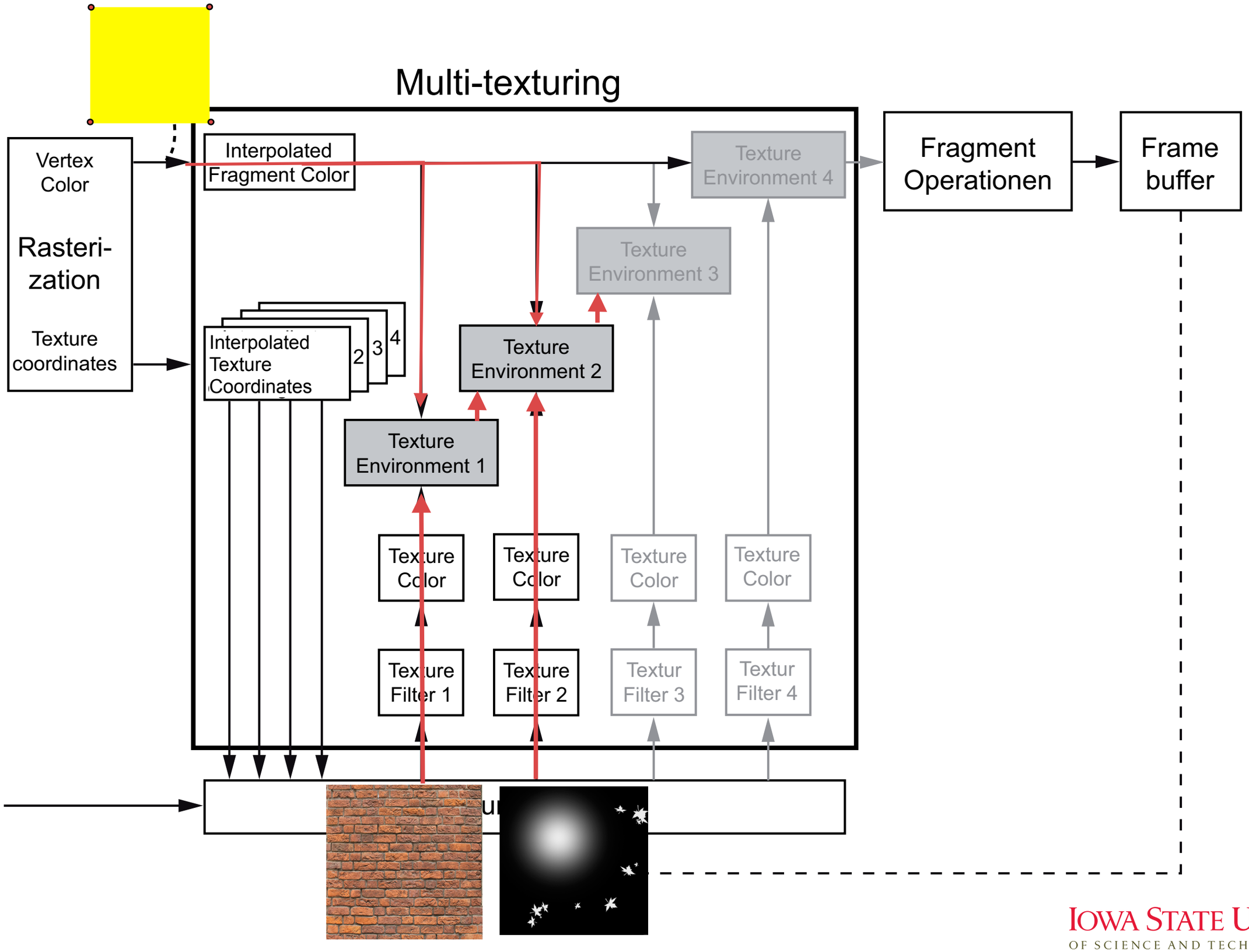
Quad

Texture 0

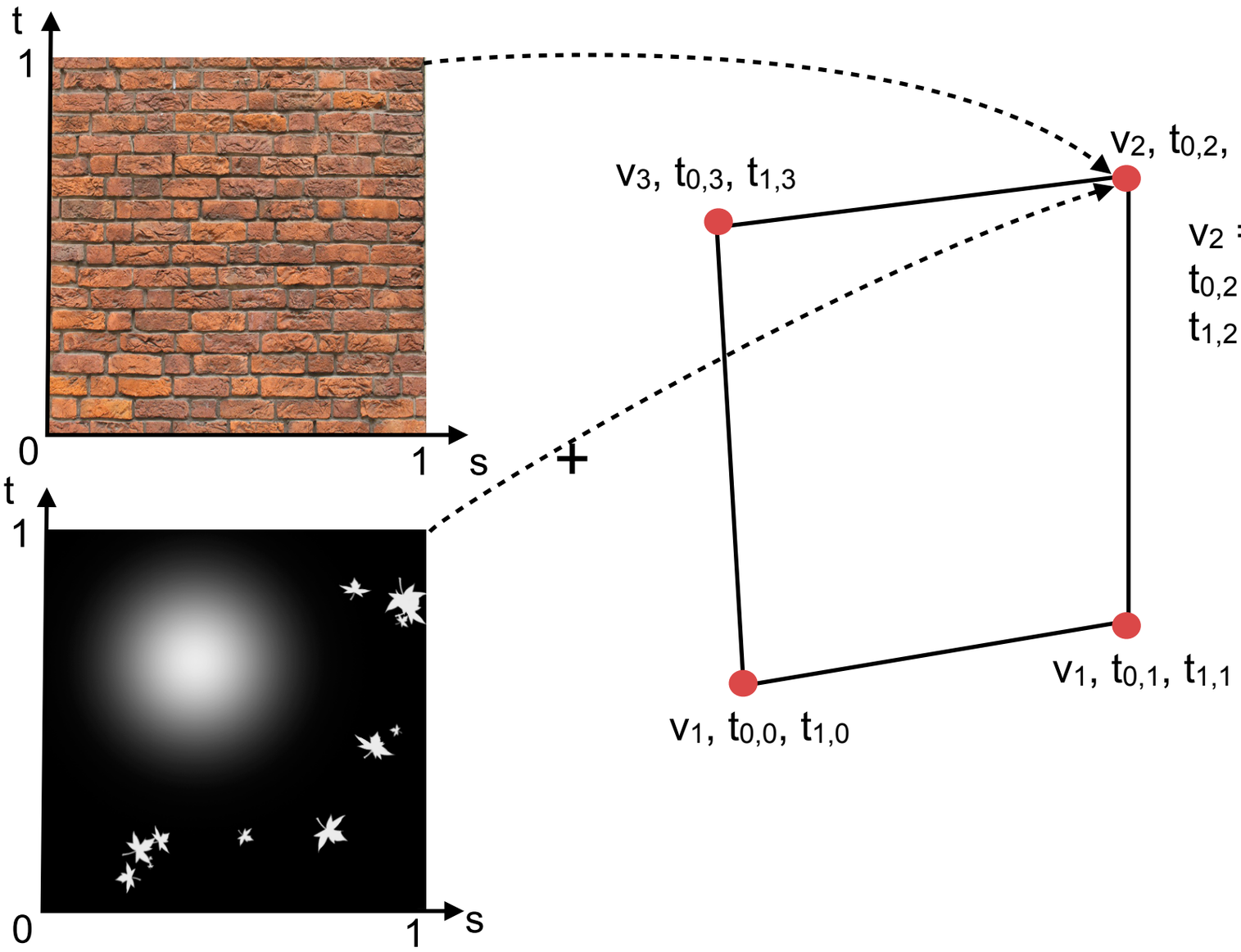


Texture 1

Example

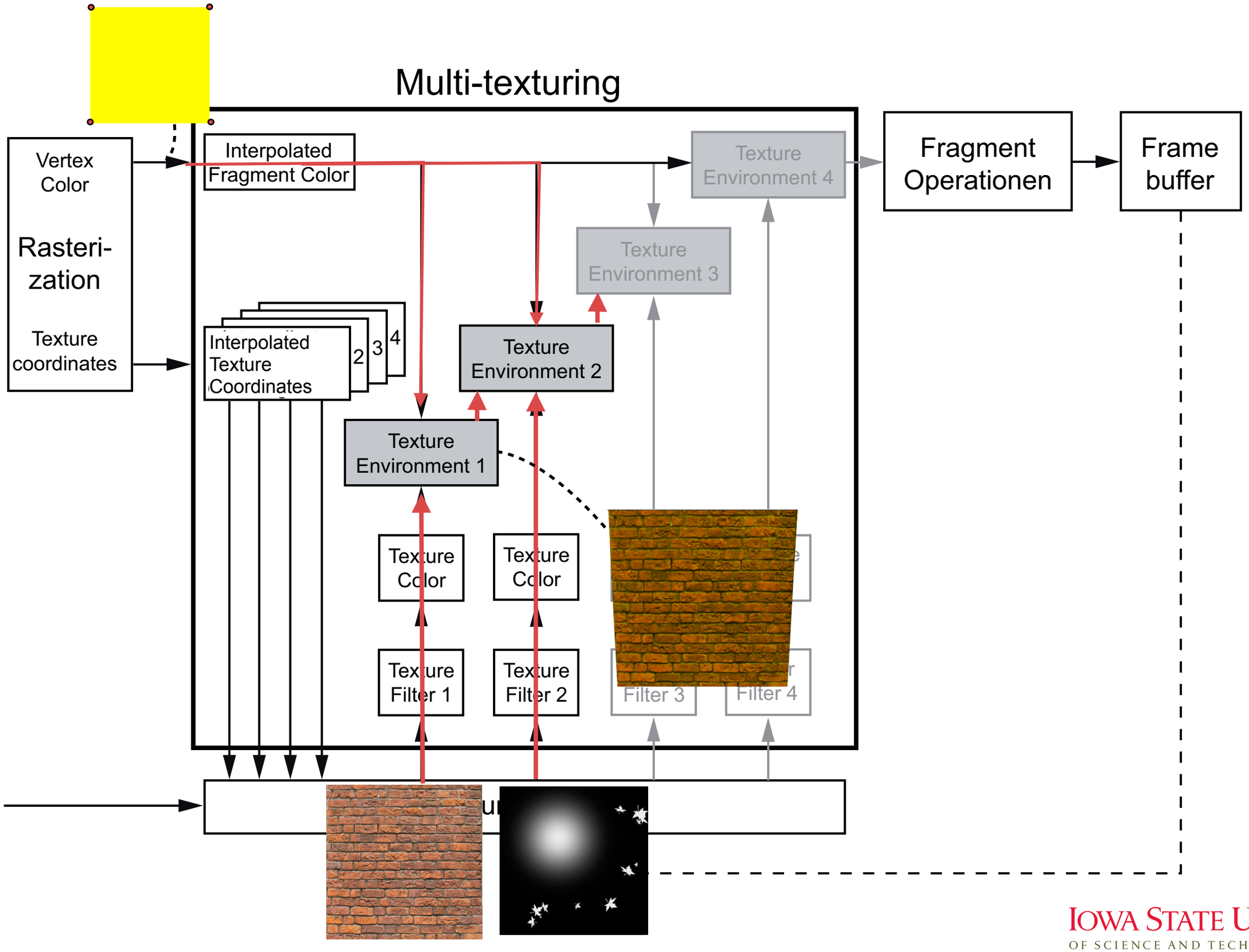

Multi-texturing
Vertex Color
Rasteri-zation
Texture coordinates
Interpolated Fragment Color
Interpolated Texture Coordinates
2
3
4
Texture Environment 4
Texture Environment 3
Texture Environment 2
Texture Environment 1
Texture Color
Texture Color
Texture Color
Texture Color
Texture Filter 1
Texture Filter 2
Textur Filter 3
Textur Filter 4
Fragment Operationen
Frame buffer

Multi-Texture Coordinates



- Texture coordinates describe the association between a primitive and the texture.
- The coordinates belong to primitive vertices; every vertex gets its own texture coordinates.
- For multi-texturing, each texture must have it's own coordinates
- The texture coordinates can be different for all units

Example


Multi-texturing
Vertex Color
Rasteri-zation
Texture coordinates
Interpolated Fragment Color
Interpolated Texture Coordinates
2
3
4
Texture Environment 4
Texture Environment 3
Texture Environment 2
Texture Environment 1
Texture Color
Texture Color
Texture Filter 1
Texture Filter 2
Filter 3
Filter 4
Fragment Operationen
Frame buffer

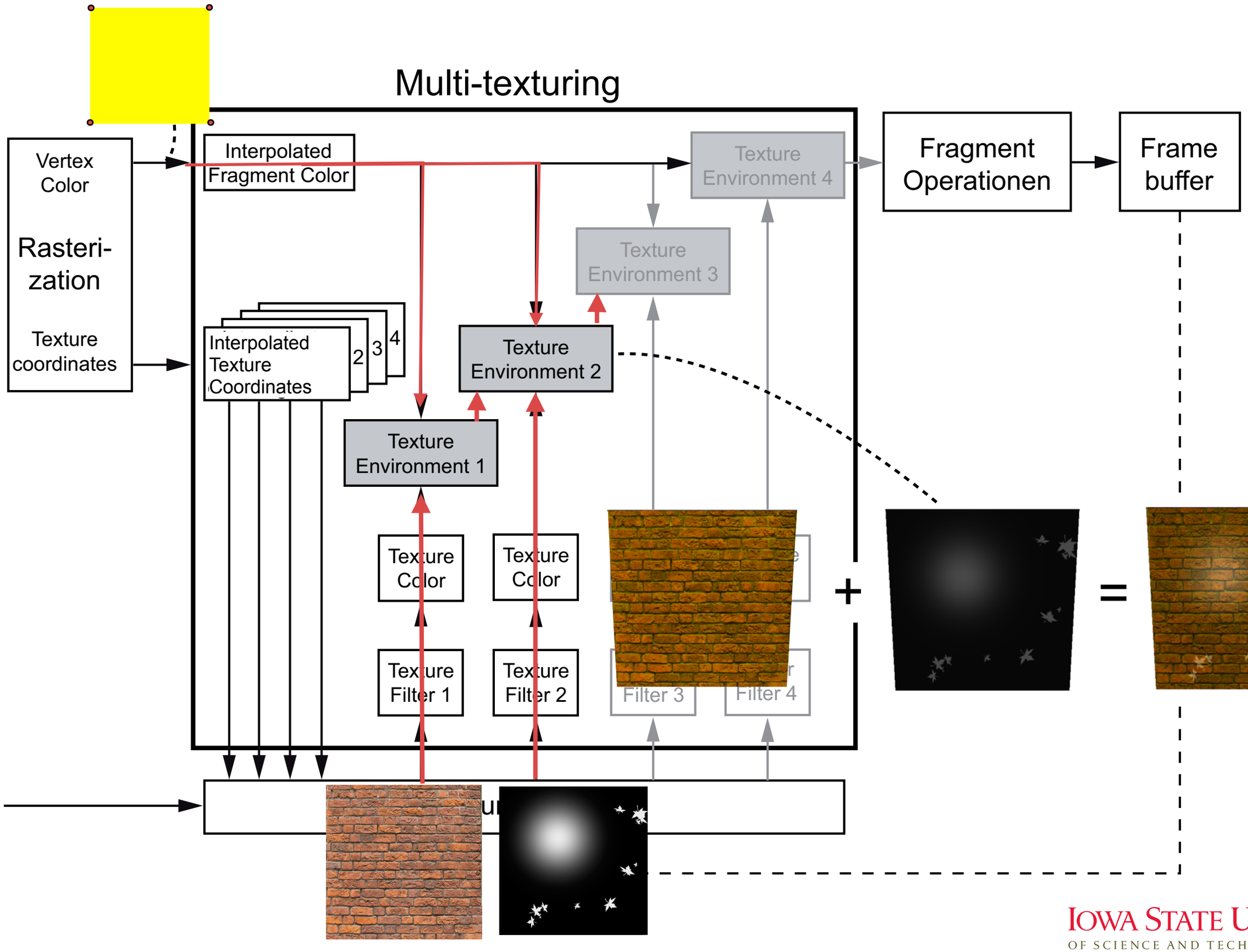
Example


```
[The code to load a texture from file is missing in this example]

//********************************************************************************
  //********************************************************************************
  // Create the first texture

  glActiveTexture(GL_TEXTURE0);

  // Generate a texture, this function allocates the memory and
  // associates the texture with a variable.
  glGenTextures(0, &texture0 );

  // Set a texture as active texture.
glBindTexture( GL_TEXTURE_2D, texture0 );

// Change the parameters of your texture units.
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 );

//********************************************************************************
// Create a texture and load it to your graphics hardware. This texture is automatically associated
// with texture 0 and the textuer variable "texture" / the active texture.
if(channels0 == 3)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width0, height0, 0, GL_BGR, GL_UNSIGNED_BYTE, data0);
else if(channels0 == 4)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width0, height0, 0, GL_BGRA, GL_UNSIGNED_BYTE, data0);
```

Enable the texture capability for both texture units

Every function that belongs to texturing affects the activated texture unit after this call.

Example


Multi-texturing
Vertex Color
Rasteri-zation
Texture coordinates
Interpolated Fragment Color
Interpolated Texture Coordinates
2
3
4
Texture Environment 4
Texture Environment 3
Texture Environment 2
Texture Environment 1
Texture Color
Texture Color
Texture Filter 1
Texture Filter 2
Filter 3
Filter 4
Fragment Operationen
Frame buffer
+
=

Example


```
//********************************                    ****************************
//********************************                    ****************************
// Create second texture

glActiveTexture(GL_TEXTURE1);

// Generate a texture, this function allocates the memory and
// associates the texture with a variable.
glGenTextures(1, &texture1 );

// Set a texture as active texture.
glBindTexture( GL_TEXTURE_2D, texture1 );
```

Enable the texture capability for both texture units

Every function that belongs to texturing affects the activated texture unit after this call.

```
// Change the parameters of your texture units.
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 );

//*************************************************************************
// Create a texture and load it to your graphics hardware. This texture is automatically associated
// with texture 0 and the textuer variable "texture" / the active texture.
if(channels1 == 3)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width1, height1, 0, GL_BGR, GL_UNSIGNED_BYTE, data1);
else if(channels1 == 4)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width1, height1, 0, GL_BGRA, GL_UNSIGNED_BYTE, data1);
```

Example

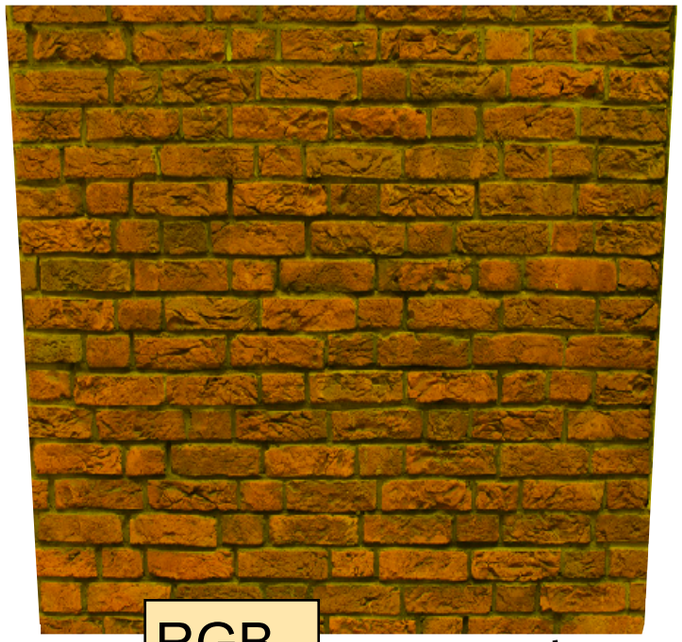
When you create uniform variables, make sure that the right texture unit is active and the texture is correct texture is bind to this unit.

```
// This binds the first texture
int texLoc = glGetUniformLocation(ShaderID, "BaseTex");

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, _background_texture);
glUniform1i(texLoc, 0);
```

```
[...]

// This binds the second texture
texLoc = glGetUniformLocation(ShaderID, "ReflectTex");

glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, _light_texture);
glUniform1i(texLoc, 1);
```

Mix with DECAL mode

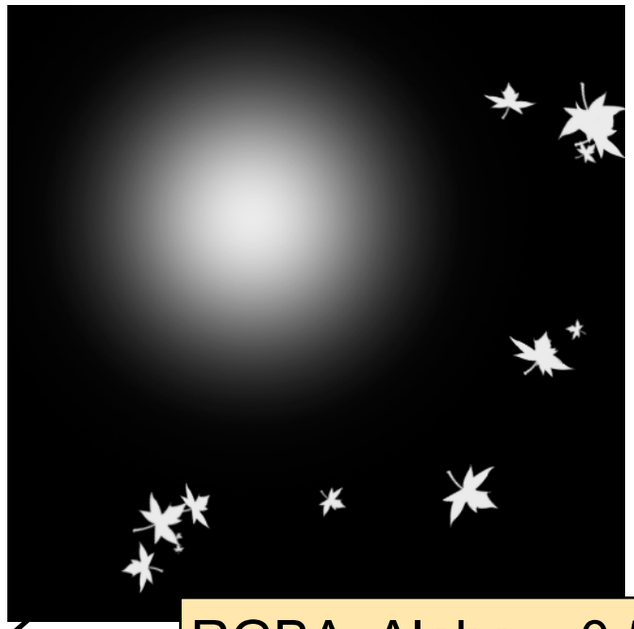
Output from texture unit 0



RGB

+

Source texture of texture unit 1



RGBA, Alpha = 0.5

=

Output

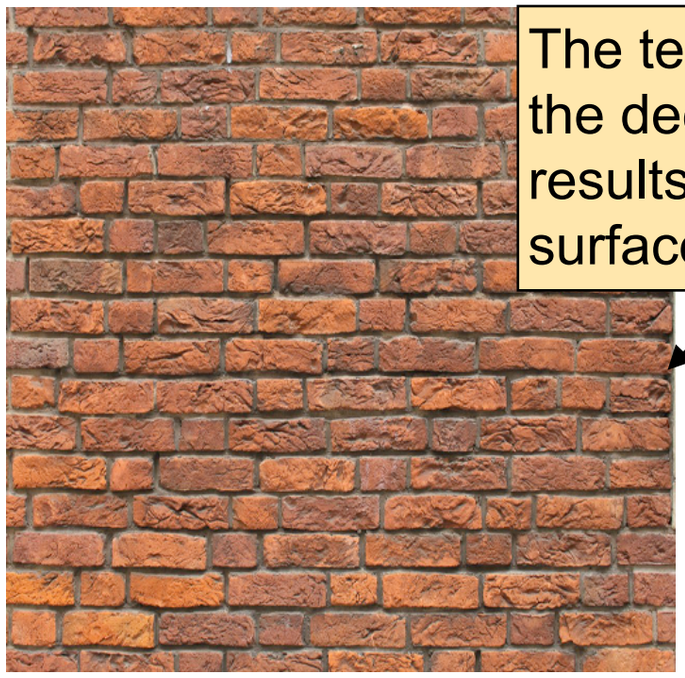
$$\begin{bmatrix} (1-A_s)R_p + R_s A_s \\ (1-G_s)G_p + G_s A_s \\ (1-A_s)B_p + B_s A_s \\ A_s \cdot A_p \end{bmatrix} = \begin{bmatrix} rR \\ rG \\ rB \\ rA \end{bmatrix}$$

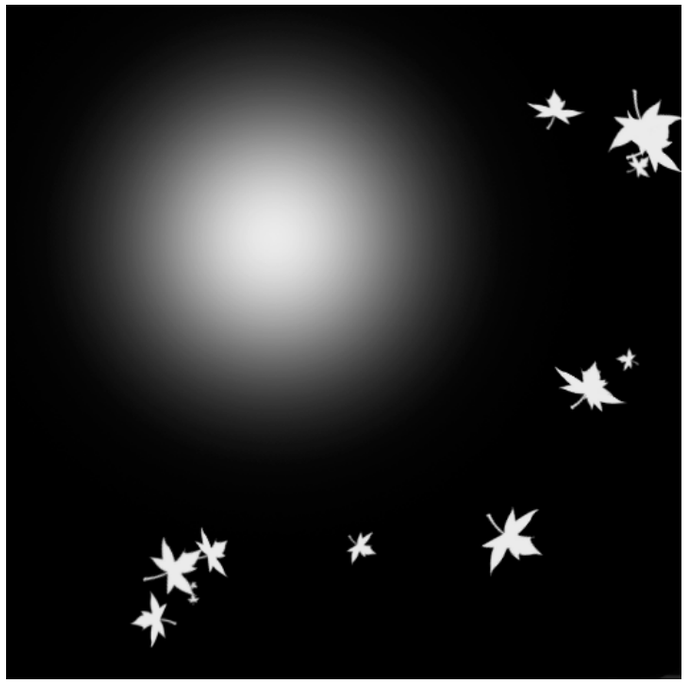
Remember: the applied equation depends on the internal color format of the source texture

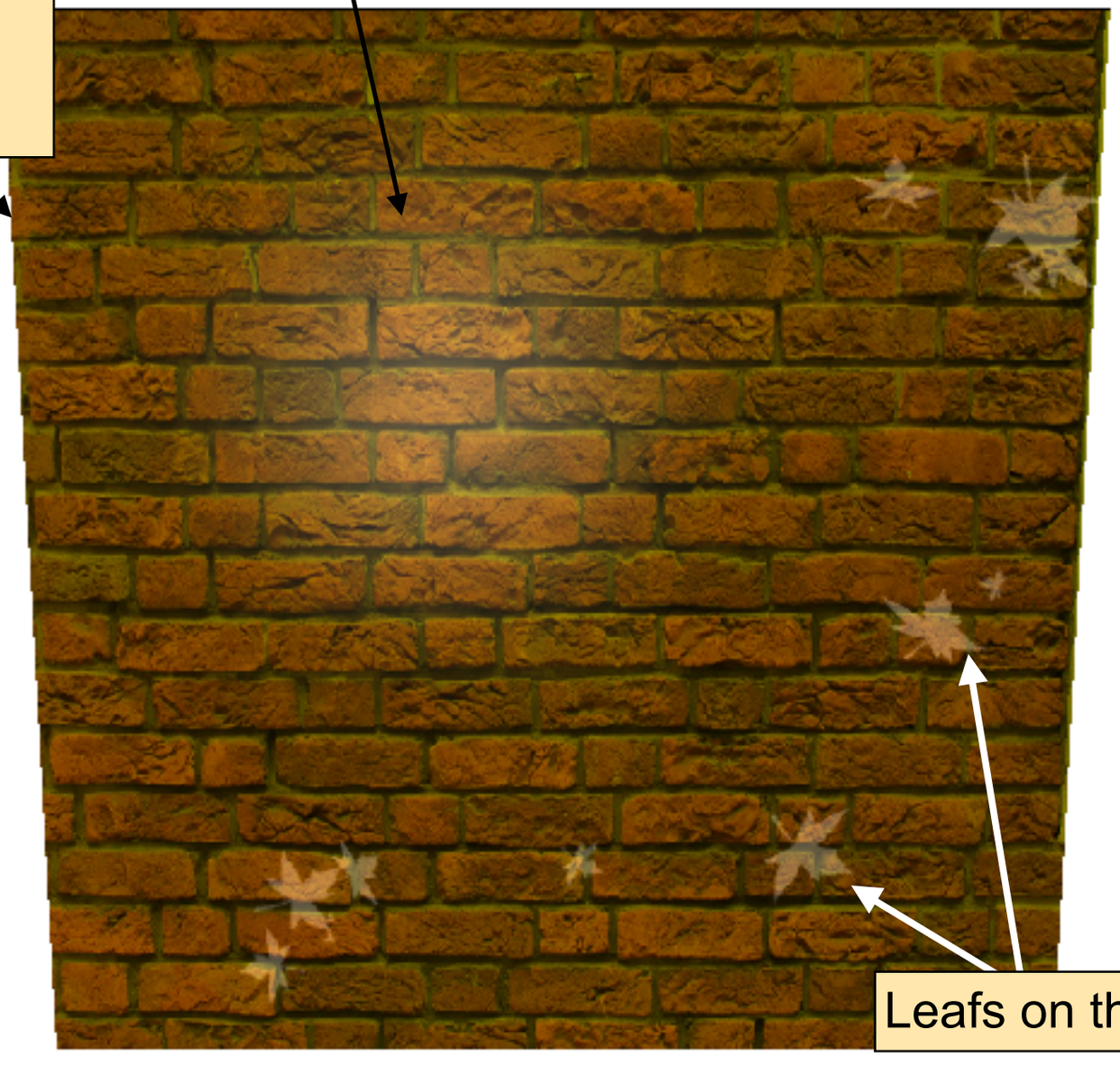
Equation for internal format: GL_RGBA

- R,G,B,A: the color components
- subscript p: previous texture unit
- subscript s: source texture of this unit

Result






The term (1-As) of the decal function results in a darker surface
Light on the wall
Leafs on the wall

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