UJIAN TENGAH SEMESTER (UTS)

GRAFIKA KOMPUTER

Iola Sava Lintang Ariella Mahesti

240533602799

(PTI B)

UNIVERSITAS NEGERI MALANG

FAKULTAS TEKNIK

PROGRAM STUDI PENDIDIKAN TEKNIK INFORMATIKA

2026

1. Source Code :

```
stbi_set_flip_vertically_on_load(true);
    unsigned char *data = stbi_load("profile.jpg", &width,
&height, &nrChannels, 0);
    if (data)
    {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
GL_RGB, GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);
    }
    else
    {
        std::cout << "Failed to load texture" << std::endl;
    }
    stbi_image_free(data);
    // texture 2
    // ---------
    glGenTextures(1, &texture2);
    glBindTexture(GL_TEXTURE_2D, texture2);
    // 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);
    // set texture filtering parameters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    // load image, create texture and generate mipmaps
    data = stbi_load("nim.png", &width, &height, &nrChannels, 0);
    if (data)
    {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
GL_RGB, GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);
    }
    else
    {
        std::cout << "Failed to load texture nim.png" <<
std::endl;
```

```
    }
    stbi_image_free(data);
```

Penjelasan :

- stbi_set_flip_vertically_on_load(true): Digunakan karena sumbu Y pada koordinat tekstur OpenGL dimulai dari bawah ke atas, sedangkan gambar digital umumnya dibaca dari atas ke bawah.
- stbi_load: Fungsi dari library stb_image untuk membaca file gambar dan mengambil informasi lebar (width), tinggi (height), serta jumlah channel warna (nrChannels).
- glTexImage2D: Berfungsi untuk generate tekstur di dalam OpenGL. Parameter format warna sangat penting; karena file gambar NIM terdeteksi hanya memiliki 3 channel warna, maka parameter internal dan format data harus diset sebagai GL_RGB, bukan GL_RGBA.
- glGenerateMipmap: Menghasilkan kumpulan gambar beresolusi lebih rendah secara otomatis untuk optimasi performa saat objek berada jauh dari kamera.

Source Code :

```
    ourShader.use();
    ourShader.setInt("texture1", 0);
    ourShader.setInt("texture2", 1);
```

Penjelasan :

Kode ini menghubungkan texture samplers yang ada di Fragment Shader (texture1 dan texture2) dengan Texture Unit yang ada di OpenGL (GL_TEXTURE0 dan GL_TEXTURE1). Dan diatur menggunakan fungsi setInt untuk mengirimkan nilai integer ke lokasi variabel uniform di shader.

Source Code :

```
glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, texture1);
        glActiveTexture(GL_TEXTURE1);
        glBindTexture(GL_TEXTURE_2D, texture2);

        glBindVertexArray(VAO);
        glDrawArrays(GL_TRIANGLES, 0, 36);
```

Penjelasan :

Sebelum menggambar objek menggunakan glDrawArrays, program harus mengaktifkan Texture Unit menggunakan glActiveTexture dan mengikat (bind) ID tekstur yang sudah dibuat sebelumnya menggunakan glBindTexture. Karena menggunakan lebih dari satu tekstur, kedua tekstur harus diikat ke unit yang berbeda (TEXTURE0 dan TEXTURE1).

2. Source Code :

```
#version 330 core
out vec4 FragColor;

in vec2 TexCoord; // Harus sama dengan 'out' di vertex shader

uniform sampler2D texture1;
uniform sampler2D texture2;

void main() {
    // Gunakan nilai tetap (misal 0.2) jika tidak mengirim
mixValue dari main.cpp
    FragColor = mix(texture(texture1, TexCoord), texture(texture2,
vec2(TexCoord.x, TexCoord.y)), 0.9);
}
```

Penjelasan :

Pada Fragment Shader, fungsi bawaan GLSL yaitu mix() digunakan untuk mencampurkan dua buah nilai berdasarkan parameter ketiga (faktor interpolasi). Disini saya menggunakan nilai 0.9 (angka belakang dari nim) dan menunjukkan bahwa warna output akhir (FragColor) akan mengambil 10% visibilitas dari texture1 dan 90% visibilitas dari texture2, sehingga gambar tekstur kedua akan terlihat lebih dominan.

3. Source Code :

```
while (!glfwWindowShouldClose(window))
    {
        // input
        // -----
        processInput(window);

        // render
        // ------
```

```
        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); //
also clear the depth buffer now!

        // bind textures on corresponding texture units
        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, texture1);
        glActiveTexture(GL_TEXTURE1);
        glBindTexture(GL_TEXTURE_2D, texture2);

        // activate shader
        ourShader.use();

        // create transformations
        glm::mat4 model         = glm::mat4(1.0f); // make sure to
initialize matrix to identity matrix first
        glm::mat4 view          = glm::mat4(1.0f);
        glm::mat4 projection    = glm::mat4(1.0f);
        model = glm::rotate(model, (float)glfwGetTime(),
glm::vec3(0.5f, 1.0f, 0.0f));
        view  = glm::translate(view, glm::vec3(0.0f, 0.0f,
-3.0f));
        projection = glm::perspective(glm::radians(45.0f),
(float)SCR_WIDTH / (float)SCR_HEIGHT, 0.1f, 100.0f);
        // retrieve the matrix uniform locations
        unsigned int modelLoc = glGetUniformLocation(ourShader.ID,
"model");
        unsigned int viewLoc  = glGetUniformLocation(ourShader.ID,
"view");
        // pass them to the shaders (3 different ways)
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE,
glm::value_ptr(model));
        glUniformMatrix4fv(viewLoc, 1, GL_FALSE, &view[0][0]);
        // note: currently we set the projection matrix each
frame, but since the projection matrix rarely changes it's often
best practice to set it outside the main loop only once.
        ourShader.setMat4("projection", projection);

        // render box
```

```
        glBindVertexArray(VAO);
        glDrawArrays(GL_TRIANGLES, 0, 36);


        // glfw: swap buffers and poll IO events (keys
pressed/released, mouse moved etc.)
        //
-------------------------------------------------------------------
-------------
        glfwSwapBuffers(window);
        glfwPollEvents();
    }
```

Penjelasan :

Dalam program ini adalah fungsi utama perulangan. Fungsi ini mengecek setiap awal iterasi apakah OpenGL (GLFW) telah menerima instruksi untuk menutup jendela (misalnya pengguna menekan tombol silang atau tombol ESC). Jika belum, loop akan terus berjalan (menggambar frame berikutnya).

Output :

Video Output : Output1_Iola Sava Lintang Ariella Mahesti_240533603799.mp4

Tantangan Tambahan

Source Code :

```
#version 330 core
out vec4 FragColor;

in vec2 TexCoord;

uniform sampler2D texture1;
uniform sampler2D texture2;

uniform float mixValue;

void main() {
    FragColor = mix(texture(texture1, TexCoord), texture(texture2,
TexCoord), mixValue);
}
```

Penjelasan :

Pada berkas Fragment Shader, ditambahkan sebuah variabel uniform float mixValue. Variabel uniform ini berfungsi sebagai jembatan komunikasi untuk menerima data dari aplikasi C++ (CPU) ke Shader (GPU). Fungsi bawaan GLSL yaitu mix() kemudian menggunakan variabel mixValue ini sebagai faktor interpolasi. Jika nilainya berubah, maka opacity antara texture1 dan texture2 juga akan berubah secara dinamis pada layar.

Source Code :

```
float mixValue = 0.2f;

void processInput(GLFWwindow *window)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);

    // Tekan Panah Atas untuk menambah visibilitas gambar kedua
    if (glfwGetKey(window, GLFW_KEY_UP) == GLFW_PRESS)
    {
        mixValue += 0.001f;
```

```
        if(mixValue >= 1.0f)
            mixValue = 1.0f; // Dibatasi maksimal 1.0
    }

    // Tekan Panah Bawah untuk mengurangi visibilitas gambar kedua
    if (glfwGetKey(window, GLFW_KEY_DOWN) == GLFW_PRESS)
    {
        mixValue -= 0.001f;
        if(mixValue <= 0.0f)
            mixValue = 0.0f;
    }
}
```

Penjelasan :

MixValue dideklarasikan dengan nilai awal 0.2f. Pada fungsi processInput(), ditambahkan logika menggunakan fungsi glfwGetKey() untuk mendeteksi penekanan tombol panah atas (GLFW_KEY_UP) dan panah bawah (GLFW_KEY_DOWN). Saat panah atas ditahan, nilai mixValue akan bertambah sebesar 0.001f per frame, membuat tekstur kedua semakin jelas. Sebaliknya, panah bawah akan menguranginya. Terdapat juga logika pembatasan (clamping) agar nilai mixValue tidak melebihi 1.0 atau kurang dari 0.0.

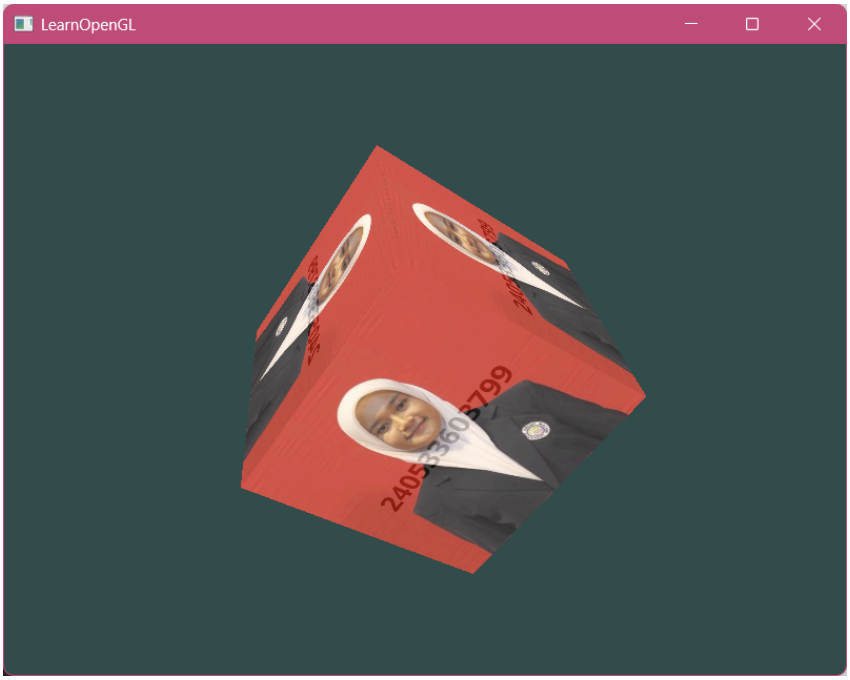
Source Code :

```
// activate shader
        ourShader.use();
        ourShader.setFloat("mixValue", mixValue);
```

Penjelasan :

Agar perubahan variabel mixValue pada memori sistem (RAM) dapat berdampak pada proses rendering, nilai tersebut harus dikirimkan ke memori kartu grafis (VRAM) di setiap iterasinya. Hal ini dilakukan di dalam Render Loop tepat setelah program shader diaktifkan (ourShader.use()). Fungsi ourShader.setFloat() dipanggil untuk mencari lokasi variabel uniform bernama "mixValue" di dalam shader, kemudian mengisinya dengan nilai mixValue terbaru yang didapatkan dari pemrosesan input. Hal ini menghasilkan efek animasi transisi (transparansi) yang halus saat tombol keyboard ditekan.

Output :



Link Video : Output 2_Iola Sava Lintang Ariella Mahesti_240533603799.mp4