Modern computer graphics introduction: A quad with a texture

Introduction

We will now use our knowledge of indices to draw a quad, and our knowledge of vertex buffers to send UV coordinates to the GPU. This will allow us to use a sampler in the fragment shader, todisplay a texture on the quad.

Note that we will use the occasion to test different kind of sampling for our texture.

Members of the quad scene

Our scene will need new members:

07.TextureQuad.cpp

```
#ifndef SCENE07TEXTUREQUAD_HPP
#define SCENE07TEXTUREQUAD_HPP

#include <SDL3/SDL_gpu.h>
#include "Scene.hpp"
#include <array>
#include <string>

using std::array;
using std::string;

class Scene07TextureQuad : public Scene {
public:
    void Load(Renderer& renderer) override;
    bool Update(float dt) override;
    void Draw(Renderer& renderer) override;
    void Unload(Renderer& renderer) override;

private:
    array<string, 6> samplerNames {
        "PointClamp",
        "PointWrap",
        "LinearClamp",
        "LinearWrap",
        "AnisotropicClamp",
        "AnisotropicWrap"
    };

    InputState inputState;
    const char* basePath {nullptr};
    SDL_GPUShader* vertexShader {nullptr};
    SDL_GPUShader* fragmentShader {nullptr};

    SDL_GPUGraphicsPipeline* pipeline {nullptr};
    SDL_GPUBuffer* vertexBuffer {nullptr};
    SDL_GPUBuffer* indexBuffer {nullptr};
    SDL_GPUTexture* texture {nullptr};
    array<SDL_GPUSampler*, 6> samplers {};
    int currentSamplerIndex {0};
};
```

```
#endif //SCENE07TEXTUREQUAD_HPP
```

Pipeline

The pipeline will be updated to include the UV coordinates.

07.TextureQuad.cpp

```
void Scene07TextureQuad::Load(Renderer& renderer) {
    basePath = SDL_GetBasePath();
    vertexShader = renderer.LoadShader(basePath, "TexturedQuad.vert", 0, 0, 0, 0);
    fragmentShader = renderer.LoadShader(basePath, "TexturedQuad.frag", 1, 0, 0, 0);

    SDL_Surface* imageData = renderer.LoadBMPImage(basePath, "ravioli.bmp", 4);
    if (imageData == nullptr) {
        SDL_Log("Could not load image data!");
    }

    // Create the pipeline
    SDL_GPUGraphicsPipelineCreateInfo pipelineCreateInfo = {
        .vertex_shader = vertexShader,
        .fragment_shader = fragmentShader,
        // This is set up to match the vertex shader layout!
        .vertex_input_state = SDL_GPUVertexInputState {
            .vertex_buffer_descriptions = new SDL_GPUVertexBufferDescription[1] {{
                .slot = 0,
                .pitch = sizeof(PositionTextureVertex),
                .input_rate = SDL_GPU_VERTEXINPUTRATE_VERTEX,
                .instance_step_rate = 0,
            }},
            .num_vertex_buffers = 1,
            .vertex_attributes = new SDL_GPUVertexAttribute[2] {{
                .location = 0,
                .buffer_slot = 0,
                .format = SDL_GPU_VERTEXELEMENTFORMAT_FLOAT3,
                .offset = 0
            }, {
                .location = 1,
                .buffer_slot = 0,
                .format = SDL_GPU_VERTEXELEMENTFORMAT_FLOAT2,
                .offset = sizeof(float) * 3
            }},
            .num_vertex_attributes = 2,
        },
        .primitive_type = SDL_GPU_PRIMITIVETYPE_TRIANGLELIST,
        .target_info = {
            .color_target_descriptions = new SDL_GPUColorTargetDescription[1] {{
               .format = SDL_GetGPUSwapchainTextureFormat(renderer.device,
        renderer.renderWindow)
            }},
            .num_color_targets = 1,
        },
    };
    pipeline = renderer.CreateGPUGraphicsPipeline(pipelineCreateInfo);

    // Clean up shader resources
    renderer.ReleaseShader(vertexShader);
    renderer.ReleaseShader(fragmentShader);
...
```

Samplers

We need to fill our sampler array in order to use them:

```
...
    // PointClamp
    samplers[0] = renderer.CreateSampler(SDL_GPUSamplerCreateInfo {
        .min_filter = SDL_GPU_FILTER_NEAREST,
        .mag_filter = SDL_GPU_FILTER_NEAREST,
        .mipmap_mode = SDL_GPU_SAMPLERMIPMAPMODE_NEAREST,
        .address_mode_u = SDL_GPU_SAMPLERADDRESSMODE_CLAMP_TO_EDGE,
        .address_mode_v = SDL_GPU_SAMPLERADDRESSMODE_CLAMP_TO_EDGE,
        .address_mode_w = SDL_GPU_SAMPLERADDRESSMODE_CLAMP_TO_EDGE,
    });
    // PointWrap
    samplers[1] = renderer.CreateSampler(SDL_GPUSamplerCreateInfo {
        .min_filter = SDL_GPU_FILTER_NEAREST,
        .mag_filter = SDL_GPU_FILTER_NEAREST,
        .mipmap_mode = SDL_GPU_SAMPLERMIPMAPMODE_NEAREST,
        .address_mode_u = SDL_GPU_SAMPLERADDRESSMODE_REPEAT,
        .address_mode_v = SDL_GPU_SAMPLERADDRESSMODE_REPEAT,
        .address_mode_w = SDL_GPU_SAMPLERADDRESSMODE_REPEAT,
    });
    // LinearClamp
    samplers[2] = renderer.CreateSampler(SDL_GPUSamplerCreateInfo {
        .min_filter = SDL_GPU_FILTER_LINEAR,
        .mag_filter = SDL_GPU_FILTER_LINEAR,
        .mipmap_mode = SDL_GPU_SAMPLERMIPMAPMODE_LINEAR,
        .address_mode_u = SDL_GPU_SAMPLERADDRESSMODE_CLAMP_TO_EDGE,
        .address_mode_v = SDL_GPU_SAMPLERADDRESSMODE_CLAMP_TO_EDGE,
        .address_mode_w = SDL_GPU_SAMPLERADDRESSMODE_CLAMP_TO_EDGE,
    });
    // LinearWrap
    samplers[3] = renderer.CreateSampler(SDL_GPUSamplerCreateInfo {
        .min_filter = SDL_GPU_FILTER_LINEAR,
        .mag_filter = SDL_GPU_FILTER_LINEAR,
        .mipmap_mode = SDL_GPU_SAMPLERMIPMAPMODE_LINEAR,
        .address_mode_u = SDL_GPU_SAMPLERADDRESSMODE_REPEAT,
        .address_mode_v = SDL_GPU_SAMPLERADDRESSMODE_REPEAT,
        .address_mode_w = SDL_GPU_SAMPLERADDRESSMODE_REPEAT,
    });
    // AnisotropicClamp
    samplers[4] = renderer.CreateSampler(SDL_GPUSamplerCreateInfo{
        .min_filter = SDL_GPU_FILTER_LINEAR,
        .mag_filter = SDL_GPU_FILTER_LINEAR,
        .mipmap_mode = SDL_GPU_SAMPLERMIPMAPMODE_LINEAR,
        .address_mode_u = SDL_GPU_SAMPLERADDRESSMODE_CLAMP_TO_EDGE,
        .address_mode_v = SDL_GPU_SAMPLERADDRESSMODE_CLAMP_TO_EDGE,
        .address_mode_w = SDL_GPU_SAMPLERADDRESSMODE_CLAMP_TO_EDGE,
        .max_anisotropy = 4,
        .enable_anisotropy = true,
    });
    // AnisotropicWrap
    samplers[5] = renderer.CreateSampler(SDL_GPUSamplerCreateInfo {
        .min_filter = SDL_GPU_FILTER_LINEAR,
        .mag_filter = SDL_GPU_FILTER_LINEAR,
        .mipmap_mode = SDL_GPU_SAMPLERMIPMAPMODE_LINEAR,
        .address_mode_u = SDL_GPU_SAMPLERADDRESSMODE_REPEAT,
        .address_mode_v = SDL_GPU_SAMPLERADDRESSMODE_REPEAT,
        .address_mode_w = SDL_GPU_SAMPLERADDRESSMODE_REPEAT,
        .max_anisotropy = 4,
        .enable_anisotropy = true,
    });
...
```

Creating the buffers

We will create the usual buffers, plus a texture to be able to upload the graphics data to the GPU.

```
...
    // Create the vertex buffer
    SDL_GPUBufferCreateInfo vertexBufferCreateInfo = {
        .usage = SDL_GPU_BUFFERUSAGE_VERTEX,
        .size = sizeof(PositionTextureVertex) * 4
    };
    vertexBuffer = renderer.CreateBuffer(vertexBufferCreateInfo);
    renderer.SetBufferName(vertexBuffer, "Ravioli Vertex Buffer");

    // Create the index buffer
    SDL_GPUBufferCreateInfo indexBufferCreateInfo = {
        .usage = SDL_GPU_BUFFERUSAGE_INDEX,
        .size = sizeof(Uint16) * 6
    };
    indexBuffer = renderer.CreateBuffer(indexBufferCreateInfo);

    // Create texture
    SDL_GPUTextureCreateInfo textureInfo {
        .type = SDL_GPU_TEXTURETYPE_2D,
        .format = SDL_GPU_TEXTUREFORMAT_R8G8B8A8_UNORM,
        .usage = SDL_GPU_TEXTUREUSAGE_SAMPLER,
        .width = static_cast<Uint32>(imageData->w),
        .height = static_cast<Uint32>(imageData->h),
        .layer_count_or_depth = 1,
        .num_levels = 1,
    };
    texture = renderer.CreateTexture(textureInfo);
    renderer.SetTextureName(texture,"Ravioli Texture");
...
```

Preparing the transfer buffers

We will now prepare the data to be transferred to the GPU. We will have a usual transfer buffer, and a new transfer buffer to upload the texture.

```
...
    // Set the transfer buffer
    SDL_GPUTransferBufferCreateInfo transferBufferCreateInfo = {
        .usage = SDL_GPU_TRANSFERBUFFERUSAGE_UPLOAD,
        .size = (sizeof(PositionTextureVertex) * 4) + (sizeof(Uint16) * 6),
    };
    SDL_GPUTransferBuffer* transferBuffer =
        renderer.CreateTransferBuffer(transferBufferCreateInfo);

    // Map the transfer buffer and fill it with data (data is bound to the transfer
        buffer)
    auto transferData = static_cast<PositionTextureVertex *>(
        renderer.MapTransferBuffer(transferBuffer, false)
    );
    transferData[0] = PositionTextureVertex { -1,  1, 0, 0, 0 };
    transferData[1] = PositionTextureVertex {  1,  1, 0, 4, 0 };
    transferData[2] = PositionTextureVertex {  1, -1, 0, 4, 4 };
    transferData[3] = PositionTextureVertex { -1, -1, 0, 0, 4 };
    auto indexData = reinterpret_cast<Uint16*>(&transferData[4]);
    indexData[0] = 0;
    indexData[1] = 1;
    indexData[2] = 2;
    indexData[3] = 0;
```

```
    indexData[4] = 2;
    indexData[5] = 3;
    renderer.UnmapTransferBuffer(transferBuffer);

    // Setup texture transfer buffer
    Uint32 bufferSize = imageData->w * imageData->h * 4;
    SDL_GPUTransferBufferCreateInfo textureTransferBufferCreateInfo {
        .usage = SDL_GPU_TRANSFERBUFFERUSAGE_UPLOAD,
        .size = bufferSize
    };
    SDL_GPUTransferBuffer* textureTransferBuffer =
        renderer.CreateTransferBuffer(textureTransferBufferCreateInfo);
    auto textureTransferData = static_cast<PositionTextureVertex *>(
        renderer.MapTransferBuffer(textureTransferBuffer, false)
    );
    std::memcpy(textureTransferData, imageData->pixels, imageData->w * imageData->h * 4);
    renderer.UnmapTransferBuffer(textureTransferBuffer);
...
```

Upload the data

Now that we have the data ready, we can upload it to the GPU.

```
...
    renderer.BeginUploadToBuffer();
    // Upload the transfer data to the vertex and index buffer
    SDL_GPUTransferBufferLocation transferVertexBufferLocation {
        .transfer_buffer = transferBuffer,
        .offset = 0
    };
    SDL_GPUBufferRegion vertexBufferRegion {
        .buffer = vertexBuffer,
        .offset = 0,
        .size = sizeof(PositionTextureVertex) * 4
    };
    SDL_GPUTransferBufferLocation transferIndexBufferLocation {
        .transfer_buffer = transferBuffer,
        .offset = sizeof(PositionTextureVertex) * 4
    };
    SDL_GPUBufferRegion indexBufferRegion {
        .buffer = indexBuffer,
        .offset = 0,
        .size = sizeof(Uint16) * 6
    };
    SDL_GPUTextureTransferInfo textureBufferLocation {
        .transfer_buffer = textureTransferBuffer,
        .offset = 0
    };
    SDL_GPUTextureRegion textureBufferRegion {
        .texture = texture,
        .w = static_cast<Uint32>(imageData->w),
        .h = static_cast<Uint32>(imageData->h),
        .d = 1
    };

    renderer.UploadToBuffer(transferVertexBufferLocation, vertexBufferRegion, false);
    renderer.UploadToBuffer(transferIndexBufferLocation, indexBufferRegion, false);
    renderer.UploadToTexture(textureBufferLocation, textureBufferRegion, false);
    renderer.EndUploadToBuffer(transferBuffer);
    renderer.ReleaseTransferBuffer(textureTransferBuffer);
    renderer.ReleaseSurface(imageData);

    // Finally, print instructions!
```

```
    SDL_Log("Press Left/Right to switch between sampler states");
    SDL_Log("Setting sampler state to: %s", samplerNames[0].c_str());
}
```

Changing the sampler

The Update function will allow to change the sampler:

```
bool Scene07TextureQuad::Update(float dt) {
    const bool isRunning = ManageInput(inputState);

    if (inputState.IsPressed(DirectionalKey::Left))
    {
        currentSamplerIndex -= 1;
        if (currentSamplerIndex < 0)
        {
            currentSamplerIndex = samplers.size() - 1;
        }
        SDL_Log("Setting sampler state to: %s",
         samplerNames[currentSamplerIndex].c_str());
    }

    if (inputState.IsPressed(DirectionalKey::Right))
    {
        currentSamplerIndex = (currentSamplerIndex + 1) % samplers.size();
        SDL_Log("Setting sampler state to: %s",
         samplerNames[currentSamplerIndex].c_str());
    }

    return isRunning;
}
```

Drawing the quad with the texture

As with the previous buffers, we need to bind the texture before drawing.

```
void Scene07TextureQuad::Draw(Renderer& renderer) {
    renderer.Begin();

    renderer.BindGraphicsPipeline(pipeline);
    SDL_GPUBufferBinding vertexBindings { .buffer = vertexBuffer, .offset = 0 };
    renderer.BindVertexBuffers(0, vertexBindings, 1);
    SDL_GPUBufferBinding indexBindings { .buffer = indexBuffer, .offset = 0 };
    renderer.BindIndexBuffer(indexBindings, SDL_GPU_INDEXELEMENTSIZE_16BIT);

    SDL_GPUTextureSamplerBinding textureSamplerBinding {
        .texture = texture,
        .sampler = samplers[currentSamplerIndex]
    };
    renderer.BindFragmentSamplers(0, textureSamplerBinding, 1);

    renderer.DrawIndexedPrimitives(6, 1, 0, 0, 0);
    renderer.End();
}
```

Unloading the scene

Let’s clean everything!

```
void Scene07TextureQuad::Unload(Renderer& renderer) {
    for (int i = 0; i < SDL_arraysize(samplers); i += 1)
    {
```

```
        renderer.ReleaseSampler(samplers[i]);
    }
    currentSamplerIndex = 0;
    renderer.ReleaseBuffer(vertexBuffer);
    renderer.ReleaseBuffer(indexBuffer);
    renderer.ReleaseTexture(texture);
    renderer.ReleaseGraphicsPipeline(pipeline);
}
```

Updating the renderer

We need to update the renderer to include the new functions.

Renderer.cpp

```
SDL_Surface* Renderer::LoadBMPImage(const char* basePath, const char* imageFilename, int
        desiredChannels) {
    char fullPath[256];
    SDL_PixelFormat format;
    SDL_snprintf(fullPath, sizeof(fullPath), "%sContent/Images/%s", basePath,
        imageFilename);

    SDL_Surface* result = SDL_LoadBMP(fullPath);
    if (result == nullptr) {
        SDL_Log("Failed to load BMP: %s", SDL_GetError());
        return nullptr;
    }

    if (desiredChannels == 4) {
        format = SDL_PIXELFORMAT_ABGR8888;
    }
    else {
        SDL_assert(!"Unexpected desiredChannels");
        SDL_DestroySurface(result);
        return nullptr;
    }

    if (result->format != format) {
        SDL_Surface *next = SDL_ConvertSurface(result, format);
        SDL_DestroySurface(result);
        result = next;
    }

    return result;
}

SDL_GPUSampler* Renderer::CreateSampler(const SDL_GPUSamplerCreateInfo& createInfo) const
        {
    return SDL_CreateGPUSampler(device, &createInfo);
}

void Renderer::ReleaseSurface(SDL_Surface* surface) const {
    SDL_DestroySurface(surface);
}

void Renderer::SetBufferName(SDL_GPUBuffer* buffer, const string& name) const {
    SDL_SetGPUBufferName(device, buffer, name.c_str());
}

SDL_GPUTexture* Renderer::CreateTexture(const SDL_GPUTextureCreateInfo& createInfo) const
        {
    return SDL_CreateGPUTexture(device, &createInfo);
}
```

```
void Renderer::SetTextureName(SDL_GPUTexture* texture, const string& name) const {
    SDL_SetGPUTextureName(device, texture, name.c_str());
}

void Renderer::ReleaseTexture(SDL_GPUTexture* texture) const {
    SDL_ReleaseGPUTexture(device, texture);
}

void Renderer::ReleaseSampler(SDL_GPUSampler* sampler) const {
    SDL_ReleaseGPUSampler(device, sampler);
}

void Renderer::UploadToTexture(const SDL_GPUTextureTransferInfo& source, const
        SDL_GPUTextureRegion& destination,
    bool cycle) const {
    SDL_UploadToGPUTexture(copyPass, &source, &destination, cycle);
}

void Renderer::BindFragmentSamplers(Uint32 firstSlot, const SDL_GPUTextureSamplerBinding&
        bindings,
    Uint32 numBindings) const {
    SDL_BindGPUFragmentSamplers(renderPass, firstSlot, &bindings, numBindings);
}
```

Exercises

1. Change the texture.
2. Set the sampler and uv to repeat, so that the texture tiles 3 times.