Modern computer graphics introduction: Culling modes with multiple pipelines

Introduction

Triangle culling is the optimization that consist in not drawing the faces of a triangle that are not visible. This is done by checking the winding order of the triangle (Clockwise or Counter-Clockwise, CW or CCW) and deciding if we want to draw the front face, the back face or both.

We will use this case to introduce the concept of multiple pipelines. We will create six pipelines, one for each culling mode (CW no culling, CW front culling, CW back culling, then the same for CCW), and we will switch between them using the input state. We will draw two triangles, one for each culling mode.

We will use two vertex buffers, one for each triangle. The vertex buffer will contain the position and color of each vertex.

Defining the pipelines

First, we need to create the necessary data in our scene.

Scene04TriangleCullModes.hpp

```
#ifndef SCENE04TRIANGLECULLMODES_HPP
#define SCENE04TRIANGLECULLMODES_HPP

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

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

class Scene04TriangleCullModes : public Scene {
public:
    void Load(Renderer& renderer) override;
    bool Update(float dt) override;
    void Draw(Renderer& renderer) override;
    void Unload(Renderer& renderer) override;

private:
    InputState inputState;
    const char* basePath;
    SDL_GPUShader* vertexShader;
    SDL_GPUShader* fragmentShader;

    SDL_GPUBuffer* vertexBufferCW;
    SDL_GPUBuffer* vertexBufferCCW;
    int currentMode { 0 };
    array<string, 6> modeNames {
        "CW_CullNone",
        "CW_CullFront",
        "CW_CullBack",
        "CCW_CullNone",
```

```
        "CCW_CullFront",
        "CCW_CullBack"
    };
    array<SDL_GPUGraphicsPipeline*, 6> pipelines;
};

#endif //SCENE04TRIANGLECULLMODES_HPP
```

Now we can create the pipelines in the `Load` method.

Scene04TriangleCullModes.cpp

```
void Scene04TriangleCullModes::Load(Renderer& renderer) {
    basePath = SDL_GetBasePath();
    vertexShader = renderer.LoadShader(basePath, "PositionColor.vert", 0, 0, 0, 0);
    fragmentShader = renderer.LoadShader(basePath, "SolidColor.frag", 0, 0, 0, 0);

    // 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(PositionColorVertex),
                .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_UBYTE4_NORM,
                .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,
        },
    };

    for (int i = 0; i < pipelines.size(); ++i)
    {
        pipelineCreateInfo.rasterizer_state.cull_mode = (SDL_GPUCullMode) (i % 3);
        pipelineCreateInfo.rasterizer_state.front_face = (i > 2) ?
            SDL_GPU_FRONTFACE_CLOCKWISE :
            SDL_GPU_FRONTFACE_COUNTER_CLOCKWISE;

        pipelines[i] = renderer.CreateGPUGraphicsPipeline(pipelineCreateInfo);
        if (pipelines[i] == nullptr)
        {
            SDL_Log("Failed to create pipeline!");
```

```
            return;
        }
    }

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

As you can see, we add information about the culing mode in the rasterizer state info. We generate one pipeline for each combination.

Creating and uploading the vertex buffers

Now, in the same function, we can create the vertex buffers and upload data.

```
...
    // Create the vertex buffer
    SDL_GPUBufferCreateInfo vertexBufferCreateInfo = {
        .usage = SDL_GPU_BUFFERUSAGE_VERTEX,
        .size = sizeof(PositionColorVertex) * 3
    };
    vertexBufferCW = renderer.CreateBuffer(vertexBufferCreateInfo);
    vertexBufferCCW = renderer.CreateBuffer(vertexBufferCreateInfo);

    // To get data into the vertex buffer, we have to use a transfer buffer
    SDL_GPUTransferBufferCreateInfo transferBufferCreateInfo = {
        .usage = SDL_GPU_TRANSFERBUFFERUSAGE_UPLOAD,
        .size = sizeof(PositionColorVertex) * 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<PositionColorVertex *>(
        renderer.MapTransferBuffer(transferBuffer, false)
    );
    transferData[0] = PositionColorVertex {    -1,    -1, 0, 255,   0,   0, 255 };
    transferData[1] = PositionColorVertex {     1,    -1, 0,   0, 255,   0, 255 };
    transferData[2] = PositionColorVertex {     0,     1, 0,   0,   0, 255, 255 };
    transferData[3] = PositionColorVertex {     0,     1, 0, 255,   0,   0, 255 };
    transferData[4] = PositionColorVertex {     1,    -1, 0,   0, 255,   0, 255 };
    transferData[5] = PositionColorVertex {    -1,    -1, 0,   0,   0, 255, 255 };
    renderer.UnmapTransferBuffer(transferBuffer);

    // Upload the transfer data to the vertex buffer
    SDL_GPUTransferBufferLocation transferBufferLocationCW = {
        .transfer_buffer = transferBuffer,
        .offset = 0
    };
    SDL_GPUTransferBufferLocation transferBufferLocationCCW = {
            .transfer_buffer = transferBuffer,
            .offset = sizeof (PositionColorVertex) * 3
    };
    SDL_GPUBufferRegion vertexBufferRegionCW = {
        .buffer = vertexBufferCW,
        .offset = 0,
        .size = sizeof(PositionColorVertex) * 3
    };
    SDL_GPUBufferRegion vertexBufferRegionCCW = {
            .buffer = vertexBufferCCW,
            .offset = 0,
```

```
            .size = sizeof(PositionColorVertex) * 3
    };

    renderer.BeginUploadToBuffer();
    renderer.UploadToBuffer(transferBufferLocationCW, vertexBufferRegionCW, false);
    renderer.UploadToBuffer(transferBufferLocationCCW, vertexBufferRegionCCW, false);
    renderer.EndUploadToBuffer(transferBuffer);

    // Finally, print instructions!
    SDL_Log("Press Left/Right to switch between modes");
    SDL_Log("Current Mode: %s", modeNames[0].c_str());
```

See how we use only one transfer buffer to upload data to both vertex buffers. The transfer buffer location and the buffer region define where this data is taken and sent. We use the same data for both triangles.

Drawing the triangles and usual operations

In the Update function, we will use the arrow keys in order to switch culling modes. Then we will draw the two triangles with our two vertex buffers.

Do not forget to release all pipelines in the Unload function.

```
bool Scene04TriangleCullModes::Update(float dt) {
    const bool isRunning = ManageInput(inputState);

    if (inputState.IsPressed(DirectionalKey::Left))
    {
        currentMode -= 1;
        if (currentMode < 0)
        {
            currentMode = pipelines.size() - 1;
        }
        SDL_Log("Current Mode: %s", modeNames[currentMode].c_str());
    }

    if (inputState.IsPressed(DirectionalKey::Right))
    {
        currentMode = (currentMode + 1) % pipelines.size();
        SDL_Log("Current Mode: %s", modeNames[currentMode].c_str());
    }

    return isRunning;
}

void Scene04TriangleCullModes::Draw(Renderer& renderer) {
    renderer.Begin();

    renderer.BindGraphicsPipeline(pipelines[currentMode]);

    renderer.SetViewport({ 0, 0, 320, 480 });
    SDL_GPUBufferBinding vertexBindingsCW = { .buffer = vertexBufferCW, .offset = 0 };
    renderer.BindVertexBuffers(0, vertexBindingsCW, 1);
    renderer.DrawPrimitives(3, 1, 0, 0);

    renderer.SetViewport({ 320, 0, 320, 480 });
    SDL_GPUBufferBinding vertexBindingsCCW = { .buffer = vertexBufferCCW, .offset = 0 };
    renderer.BindVertexBuffers(0, vertexBindingsCCW, 1);
    renderer.DrawPrimitives(3, 1, 0, 0);

    renderer.End();
}
```

```
void Scene04TriangleCullModes::Unload(Renderer& renderer) {
    renderer.ReleaseBuffer(vertexBufferCW);
    renderer.ReleaseBuffer(vertexBufferCCW);
    for (int i = 0; i < pipelines.size(); ++i) {
        renderer.ReleaseGraphicsPipeline(pipelines[i]);
    }
}
```

That’s it! This lesson is very short, but it empowers you with the ability of using multiple pipelines in your application. This is a very powerful feature that allows you to switch between different rendering modes in a very efficient way.