Modern computer graphics introduction: Drawing with Indices

Introduction

Indexing is a technique used to reduce the amount of data that needs to be sent to the GPU. It is used to avoid redundancy in the vertex data. In this lesson, we will learn how to use indices to draw a triangle.

Using indices

Indices, as with OpenGL, are used to reference the vertices in the vertex buffer. The indices are used to define the order in which the vertices are drawn. This allows us to reuse vertices in the vertex buffer, which can be useful when drawing complex shapes.

To use indices, we need to create an index buffer. This buffer will contain the indices that reference the vertices in the vertex buffer. We can then use the index buffer to draw the vertices in the correct order.

Thus, we need an index buffer in our scene:

Scene06TriangleInstanced.h

```
#ifndef SCENE06TRIANGLEINDEXED_HPP
#define SCENE06TRIANGLEINDEXED_HPP

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

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

class Scene06TriangleIndexed : 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_GPUGraphicsPipeline* pipeline;
    SDL_GPUBuffer* vertexBuffer;
    SDL_GPUBuffer* indexBuffer;
    bool useIndexBuffer { true };
};

#endif //SCENE06TRIANGLEINDEXED_HPP
```

The index buffer will be created in the `Load` method. We will then use our now well-known transfer buffer to upload both the vertex buffer and the index buffer to the

GPU.

The pipeline creation is similar:

Scene06TriangleIndexed.cpp

```
void Scene06TriangleIndexed::Load(Renderer& renderer) {
    basePath = SDL_GetBasePath();
    vertexShader = renderer.LoadShader(basePath, "PositionColorInstanced.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,
        },
    };

    pipeline = renderer.CreateGPUGraphicsPipeline(pipelineCreateInfo);

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

Now, let's create the vertex buffer and the index buffer, and a transfer buffer in order to upload this data.

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

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

    // Set the buffer data
    SDL_GPUTransferBufferCreateInfo transferBufferCreateInfo = {
        .usage = SDL_GPU_TRANSFERBUFFERUSAGE_UPLOAD,
        .size = (sizeof(PositionColorVertex) * 3) + (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<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 };

    Uint16* indexData = reinterpret_cast<Uint16*>(&transferData[3]);
    for (Uint16 i = 0; i < 6; i += 1) {
        indexData[i] = i;
    }
    renderer.UnmapTransferBuffer(transferBuffer);

    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(PositionColorVertex) * 3
    };
    SDL_GPUTransferBufferLocation transferIndexBufferLocation {
        .transfer_buffer = transferBuffer,
        .offset = sizeof(PositionColorVertex) * 3
    };
    SDL_GPUBufferRegion indexBufferRegion {
        .buffer = indexBuffer,
        .offset = 0,
        .size = sizeof(Uint16) * 6
    };

    renderer.UploadToBuffer(transferVertexBufferLocation, vertexBufferRegion, false);
    renderer.UploadToBuffer(transferIndexBufferLocation, indexBufferRegion, false);
    renderer.EndUploadToBuffer(transferBuffer);
}
```

We have a vertex buffer of 3 vertices and an index buffer of 6 indices. As before, the transfer buffer location and the buffer region determine the layout of the data.

In the Update function, we will allow to switch between using the index buffer and not using it. The Draw function will reflect that. Finally, we also need to release the pipeline and the buffers in the Unload function.

```
bool Scene06TriangleIndexed::Update(float dt) {
    const bool isRunning = ManageInput(inputState);

    if (inputState.IsPressed(DirectionalKey::Up))
    {
        useIndexBuffer = !useIndexBuffer;
        SDL_Log("Using index buffer: %s", useIndexBuffer ? "true" : "false");
    }

    return isRunning;
}

void Scene06TriangleIndexed::Draw(Renderer& renderer) {
    renderer.Begin();

    renderer.BindGraphicsPipeline(pipeline);

    SDL_GPUBufferBinding vertexBindings = { .buffer = vertexBuffer, .offset = 0 };
    renderer.BindVertexBuffers(0, vertexBindings, 1);
    if (useIndexBuffer) {
        SDL_GPUBufferBinding indexBindings = { .buffer = indexBuffer, .offset = 0 };
        renderer.BindIndexBuffer(indexBindings, SDL_GPU_INDEXELEMENTSIZE_16BIT);
        renderer.DrawIndexedPrimitives(3, 16, 0, 0, 0);
    } else {
        renderer.DrawPrimitives(3, 16, 0, 0);
    }

    renderer.End();
}

void Scene06TriangleIndexed::Unload(Renderer& renderer) {
    renderer.ReleaseBuffer(vertexBuffer);
    renderer.ReleaseBuffer(indexBuffer);
    renderer.ReleaseGraphicsPipeline(pipeline);
}
```

Updating the renderer

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

`Renderer.cpp`

```
void Renderer::BindIndexBuffer(const SDL_GPUBufferBinding& bindings,
        SDL_GPUIndexElementSize indexElementSize) const {
    SDL_BindGPUIndexBuffer(renderPass, &bindings, indexElementSize);
}

void Renderer::DrawIndexedPrimitives(int numIndices, int numInstances, int firstIndex,
    int vertexOffset, int firstInstance) const {
    SDL_DrawGPUIndexedPrimitives(renderPass, numIndices, numInstances, firstIndex,
        vertexOffset, firstInstance);
}
```

Exercices

1. Create and additional white triangle in the vertex buffer and offset the buffer use to draw it instead of the multicolor traingle.
2. Use an index buffer to draw a square.