郭隆邦 2023-05-20

7. WebGPU动画(uniform旋转矩阵)

本节课给大家讲解如何实现WebGPU的动画效果。

`requestAnimationFrame()`请求动画帧函数

通过HTML5的请求动画帧函数`window.requestAnimationFrame()`,辅助实现WebGPU的动画效果。

requestAnimationFrame(render)实现render函数的周期性执行，默认每秒钟执行60次，也可能低于60次。

```js
let i = 0;
// 循环动画render()
function render() {
    i+=1;
    requestAnimationFrame(render);
    console.log('执行次数'+i);
}
render();
```

gl-matrix生成旋转矩阵

在上节课2.6小节基础上，更改代码，通过gl-matrix生成一个旋转矩阵，旋转几何图形，其它代码不变。

需要旋转的矩形对应顶点数据

```js
const vertexArray = new Float32Array([
    // 三角形1
    -0.3, -0.5, 0.0,
    0.3, -0.5, 0.0,
    0.3, 0.5, 0.0,
    // 三角形2
    -0.3, -0.5, 0.0,
```

```
    0.3, 0.5, 0.0,
    -0.3, 0.5, 0.0,
]);
```

gl-matrix生成旋转矩阵,传入uniform数据缓冲区

```js
const modelMatrix = glMatrix.mat4.create();
// 绕z旋转30度
glMatrix.mat4.rotateZ(modelMatrix, modelMatrix, Math.PI/6);
// 绕z旋转90度
// glMatrix.mat4.rotateZ(modelMatrix, modelMatrix, Math.PI/2)
// 在GPU显存上创建一个uniform数据缓冲区
const modelMatrixBuffer = device.createBuffer({
    size: modelMatrix.byteLength,
    usage: GPUBufferUsage.UNIFORM | GPUBufferUsage.COPY_DST
});
device.queue.writeBuffer(modelMatrixBuffer, 0, modelMatrix);
```

命令编码器对象和绘制命令 `.draw()`

回顾下1.7节 讲解的内容：命令编码器对象。

前面给大家讲解过，通过命令编码器对象 `commandEncoder` 调用绘制命令 `.draw()` ,可以把顶点数据对应几何图形绘制出来，简单点说就是调用WebGPU API，解析顶点数据生成一张图像，显示在Cavnas画布上。

```js
// 命令编码器
const commandEncoder = device.createCommandEncoder();
// 渲染通道
const renderPass = commandEncoder.beginRenderPass({
    colorAttachments: [{
        view: context.getCurrentTexture().createView(),
        storeOp: 'store',
        loadOp: 'clear',
    }]
});
renderPass.setPipeline(pipeline);
// 顶点缓冲区数据和渲染管线shaderLocation: 0表示存储位置关联起来
renderPass.setVertexBuffer(0, vertexBuffer);
// 把绑定组里面的uniform数据传递给着色器中uniform变量
renderPass.setBindGroup(0, bindGroup);
renderPass.draw(6);// 绘制顶点数据
```

```
renderPass.end();
const commandBuffer = commandEncoder.finish();
device.queue.submit([commandBuffer]);
```

动画循环`render()`重复渲染

通过命令编码器对象，完成一次绘制draw，可以生成一张图像，如果重复执行，就可以不停的生成新的图像，每次执行生成的图像可以称为一帧图像。

```js
//渲染循环
function render() {
    // 命令编码器
    const commandEncoder = device.createCommandEncoder();
    // 渲染通道
    const renderPass = commandEncoder.beginRenderPass({
        colorAttachments: [{
            view: context.getCurrentTexture().createView(),
            storeOp: 'store',

            loadOp: 'clear',
        }]
    });
    renderPass.setPipeline(pipeline);
    // 顶点缓冲区数据和渲染管线shaderLocation: 0表示存储位置关联起来
    renderPass.setVertexBuffer(0, vertexBuffer);
    // 把绑定组里面的uniform数据传递给着色器中uniform变量
    renderPass.setBindGroup(0, bindGroup);
    renderPass.draw(6);// 绘制顶点数据
    renderPass.end();
    const commandBuffer = commandEncoder.finish();
    device.queue.submit([commandBuffer]);
    requestAnimationFrame(render);
}
render()
```

WebGPU渲染循环更新uniform矩阵

动画循环`render()`中更新uniform矩阵、重新执行draw绘制，每次执行draw的时候，都是使用新的旋转矩阵，生成一张新的图像，把不同旋转矩阵对应的一帧帧图像连起来，这样就可以产生旋转动画的视觉效果。

```js
//渲染循环
let angle = 0.0;//初始旋转角度
function render() {
    angle += 0.05;//每次渲染角度增加
    const modelMatrix = glMatrix.mat4.create();
    // 每次渲染，生成新的旋转矩阵
    glMatrix.mat4.rotateZ(modelMatrix, modelMatrix,angle);
    //模型矩阵modelMatrix重新写入uniform数据的缓冲区中
    device.queue.writeBuffer(modelMatrixBuffer, 0, modelMatrix)

    // 命令编码器
    const commandEncoder = device.createCommandEncoder();
    // 渲染通道
    const renderPass = commandEncoder.beginRenderPass({
        colorAttachments: [{
            view: context.getCurrentTexture().createView(),
            storeOp: 'store',
            loadOp: 'clear',
        }]
    });
    renderPass.setPipeline(pipeline);
    // 顶点缓冲区数据和渲染管线shaderLocation: 0表示存储位置关联起来
    renderPass.setVertexBuffer(0, vertexBuffer);
    // 把绑定组里面的uniform数据传递给着色器中uniform变量
    renderPass.setBindGroup(0, bindGroup);
    renderPass.draw(6);// 绘制顶点数据
    renderPass.end();
    const commandBuffer = commandEncoder.finish();
    device.queue.submit([commandBuffer]);

    requestAnimationFrame(render);
}
render()
```

← 6. gl-matrix生成顶点着色器的矩阵

8. 绕y轴旋转动画→