SPRAWOZDANIE

Zajęcia: Grafika Komputerowa
Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium: 10
Data: 13.05.2024
Temat: "Podstawy WebGL/GLSL"

Michał Michalik
Informatyka I stopień,
stacjonarne,
4 semestr,
Gr.3a

Zadanie 1

1. Polecenie:

Program w lab11.html pokazuje wiele ruchomych czerwonych kwadratów, które odbijają się od krawędzi płótna. Płótno wypełnia cały obszar zawartości przeglądarki internetowej. Kwadraty odpowiadają również myszą: jeśli klikniesz lewym przyciskiem myszy lub klikniesz lewym przyciskiem myszy i przeciągniesz na płótnie, cały kwadrat będzie kierowany w stronę pozycji myszy. Jeśli klikniesz lewym przyciskiem myszy, dane punktów zostaną ponownie zainicjowane, więc zaczną się od środka. Możesz wstrzymać i ponownie uruchomić animację, naciskając spację.

Kwadraty są w rzeczywistości częścią jednego prymitywu WebGL typu gl.POINTS. Każdy kwadrat odpowiada jednemu z wierzchołków pierwotnego. Oczywiście renderowanie jest wykonywane przez moduł shadera wierzchołków i moduł shadera fragmentu. Kod źródłowy shaderów jest w dwóch fałszywych „skryptach” w górnej części pliku HTML.

Będziesz modyfikował kod modułu shadera i kod JavaScript, aby zaimplementować kilka różnych stylów dla prymitywu punktu. Na przykład możliwe będzie rysowanie kwadratów w różnych kolorach, rysowanie wielokątów zamiast kwadratów i tak dalej. Użytkownik będzie kontrolował program, naciskając klawisze na klawiaturze. Do ciebie należy decyzja, których klawiszy użyć, ale proszę udokumentować interfejs w odpowiednim komentarzu do funkcji doKey() lub na górze programu.

Program ma dwie funkcje, nad którymi będziesz musiał pracować: Funkcja initGL() jest wywoływana, gdy program jest uruchamiany po raz pierwszy, a funkcje updateForFrame() i render() są wywoływane dla każdej ramki animacji. Ten sam zestaw poleceń byłby legalny we wszystkich tych poleceniach, ale initGL() jest najlepszym miejscem do ustawiania rzeczy, które nie zmienią się w trakcie działania programu, takich jak położenie zmiennych i zmiennych atrybutów w module shadera; updateForFrame() jest przeznaczony do aktualizacji zmiennych JavaScript, które zmieniają się z ramki na ramkę; a render() ma na celu wykonanie rzeczywistego rysunku WebGL ramki.

Atrybut koloru

W oryginalnej wersji programu wszystkie kwadraty są czerwone. Pierwsze ćwiczenie polega na umożliwieniu przypisania innego koloru do każdego

kwadratu. Ponieważ kwadraty są naprawdę wierzchołkami w pojedynczym prymitywie typu gl.POINTS, można użyć zmiennej atrybutu dla koloru. Atrybut może mieć inną wartość dla każdego wierzchołka.

Pierwszym zadaniem jest dodanie zmiennej kolorowej typu vec3 do modułu shadera wierzchołka i użycie wartości atrybutu do pokolorowania kwadratów. Będziesz także musiał pracować po stronie JavaScript. Będziesz potrzebował Float32Array do przechowywania wartości kolorów po stronie JavaScript, a będziesz potrzebował bufora WebGL dla tego atrybutu. Program ma już jeden atrybut, który jest używany do współrzędnych wierzchołków. Będziesz robił coś podobnego do atrybutu color (poza tym, że możesz to zrobić w initGL(), ponieważ wartości kolorów nie zmienią się po ich utworzeniu). Można użyć losowych wartości w zakresie od 0,0 do 1,0 dla składników koloru.

Po uruchomieniu wielokolorowych kwadratów powinieneś ustawić kolory jako opcjonalne. Możesz włączać i wyłączać użycie tablicy wartości atrybutów za pomocą następujących poleceń, gdzie a_color_loc to identyfikator atrybutu color w programie shader:

```
gl.enableVertexAttribArray(a_color_loc); // użyj bufora atrybutów kolorów
gl.disableVertexAttribArray(a_color_loc); // nie używaj bufora
```

Gdy tablica atrybutów jest włączona, każdy wierzchołek otrzymuje swój własny kolor z buforu atrybutów. Gdy tablica atrybutów jest wyłączona, wszystkie wierzchołki otrzymują ten sam kolor, a tę wartość można ustawić za pomocą rodziny funkcji gl.vertexAttrib*. Na przykład, aby ustawić wartość używaną, gdy tablica atrybutów kolorów jest wyłączona, można użyć:

```
gl.vertexAttrib3f(a_color_loc, 1, 0, 0); // ustaw kolor atrybutu na czerwony
```

Pozwól użytkownikowi na naciśnięcie określonego klawisza, aby włączyć lub wyłączyć losowe kolory. Program ma funkcję doKey(), która jest już skonfigurowana do reagowania na wprowadzanie z klawiatury. Będziesz dodawać do programu kilka typów interakcji z klawiaturą. Aby odpowiedzieć na klawisz, musisz znać numeryczny kod klawiszy. Funkcja doKey() wysyła kod do konsoli za każdym razem, gdy użytkownik uderza klawisz, i możesz użyć tej funkcji, aby odkryć wszystkie inne kody klawiszy, których potrzebujesz.

Styl punktów

Powinieneś dodać opcję używania stylu wyświetlania dla punktów w postaci wielokąta. Pozwól użytkownikowi wybrać styl za pomocą klawiatury; na przykład, naciskając klawisze numeryczne.

Style będą musiały zostać zaimplementowane w shaderze fragmentu, a będziesz potrzebował nowej zmiennej jednolitej, aby powiedzieć modułowi shadera fragmentu, którego stylu użyć. Dodaj jednolitą zmienną typu int do shadera fragmentu, aby kontrolować styl punktu, i dodaj kod do modułu cieniującego fragmentu, aby zaimplementować różne style. Będziesz także musiał dodać zmienną po stronie JavaScript dla lokalizacji zmiennej jednolitej, a będziesz musiał wywołać glUniform1i, gdy chcesz zmienić styl.

Na przykład, żeby narysować punkt jako dysk, odrzucając niektóre piksele:

```
float dist = distance(vec2(0.5), gl_PointCoord);
if (dist > 0.5) {
   discard;
}
```

Powinieneś również wykorzystać przezroczystość alfa w niektórych stylach. Aby umożliwić korzystanie ze składnika alpha, musisz dodać następujące linie do funkcji initGL():

```
gl.enable(gl.BLEND);
gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);
```

Dzięki tym ustawieniom wartość alfa koloru będzie używana do przezroczystości w zwykły sposób. W szczególności jeden z twoich stylów powinien pokazywać punkt jako wielokąt, który zanika z całkowicie nieprzezroczystego w środku wielokąta do całkowicie przezroczystego na krawędzi.

2. Wprowadzane dane:

```
function initGL() {
    canvas = document.getElementById('glcanvas');
    gl = canvas.getContext('webgl');
    const vertexShaderSource = document.getElementById('vertex-shader').text;
    const fragmentShaderSource = document.getElementById('fragment-shader').text;
    const vertexShader = createShader(gl, gl.VERTEX_SHADER, vertexShaderSource);
    const fragmentShader = createShader(gl, gl.FRAGMENT_SHADER,
        fragmentShaderSource);
    program = createProgram(gl, vertexShader, fragmentShader);
    a_position = gl.getAttribLocation(program, 'a_position');
    a_color = gl.getAttribLocation(program, 'a_color');
    u_resolution = gl.getUniformLocation(program, 'u_resolution');
    u_mouse = gl.getUniformLocation(program, 'u_mouse');
    u_pointStyle = gl.getUniformLocation(program, 'u_pointStyle');
    positionBuffer = gl.createBuffer();
    colorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(numSquares * 2),
        gl.STATIC_DRAW);
    gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
    const colors = new Float32Array(numSquares * 3);
    for (let i = 0; i < colors.length; i++) {
        colors[i] = Math.random();
    }
    gl.bufferData(gl.ARRAY_BUFFER, colors, gl.STATIC_DRAW);
    for (let i = 0; i < numSquares; i++) {
        squares.push({
            x: Math.random() * canvas.width,
            y: Math.random() * canvas.height,
            dx: (Math.random() - 0.5) * 2,
            dy: (Math.random() - 0.5) * 2
        });
    }
```

```
        window.addEventListener('resize', resizeCanvas);
        window.addEventListener('keydown', doKey);
        canvas.addEventListener('mousemove', handleMouseMove);
        canvas.addEventListener('mousedown', handleMouseDown);
        canvas.addEventListener('mouseup', handleMouseUp);
        resizeCanvas();
        requestAnimationFrame(render);
    }
```

```
function createShader(gl, type, source) {
    const shader = gl.createShader(type);
    gl.shaderSource(shader, source);
    gl.compileShader(shader);
    const success = gl.getShaderParameter(shader, gl.COMPILE_STATUS);
    if (!success) {
        console.log(gl.getShaderInfoLog(shader));
        gl.deleteShader(shader);
        return null;
    }
    return shader;
}
```

```
function createProgram(gl, vertexShader, fragmentShader) {
    const program = gl.createProgram();
    gl.attachShader(program, vertexShader);
    gl.attachShader(program, fragmentShader);
    gl.linkProgram(program);
    const success = gl.getProgramParameter(program, gl.LINK_STATUS);
    if (!success) {
        console.log(gl.getProgramInfoLog(program));
        gl.deleteProgram(program);
        return null;
    }
    return program;
}
```

```
function resizeCanvas() {
    canvas.width = window.innerWidth;
    canvas.height = window.innerHeight;
    gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);
}
function doKey(event) {
    switch (event.keyCode) {
        case 32: // Space to pause/resume
            if (animationFrame) {
                cancelAnimationFrame(animationFrame);
                animationFrame = null;
            } else {
                requestAnimationFrame(render);
            }
            break;
        case 49: // 1 to switch to square style
            pointStyle = 0;
            break;
        case 50: // 2 to switch to disk style
            pointStyle = 1;
            break;
        case 51: // 3 to switch to gradient disk style
            pointStyle = 2;
            break;
        case 67: // C to toggle colors
            colorsEnabled = !colorsEnabled;
            break;
        default:
            break;
    }
}
```

3. Wykorzystane komendy:

Link git dodac

4. Wynik działania:

5. Wnioski:

Biblioteka WebGL/GLSL to narzędzie umożliwiające tworzenie grafiki w przeglądarkach internetowych, bazujące na bibliotece OpenGL. Choć oferuje szerokie możliwości, jej obsługa może być skomplikowana.