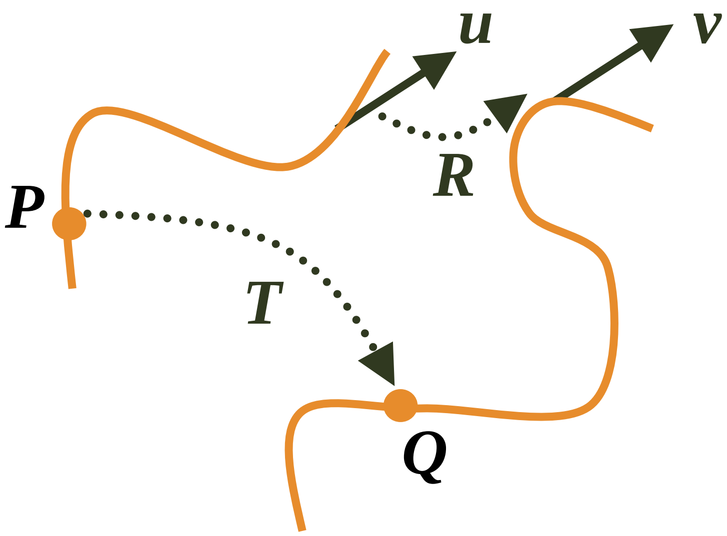
Transformations (2)

6^{TH} WEEK, 2021

Transformations

- Take a point (or vector) and map that point (or vector) into another point (or vector)



4D Column Matrices

$$Q = T(P) \longrightarrow \mathbf{q} = f(\mathbf{p})$$

Homogeneous Coordinate

$$v = R(u) \longrightarrow \mathbf{v} = f(\mathbf{u})$$

Transformation function

Affine Transformations

- Linearity – linear function

$$f(\alpha p + \beta q) = \alpha f(p) + \beta f(q)$$

- Linear transformation
  - Transforming the representation of a point (or vector) into another representation of a point (or vector)

$$\mathbf{v} = \mathbf{A}\mathbf{u}$$

$$\mathbf{A} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{u} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 0 \end{bmatrix} \qquad \mathbf{p} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ 1 \end{bmatrix}$$

4×4 Matrix Vector Point

Transformations in Homogeneous Coordinates

- Representations in homogeneous coordinates

$$Q = P + \alpha v \longrightarrow \mathbf{q} = \mathbf{p} + \alpha \mathbf{v} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 1 \end{bmatrix} + \alpha \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ 0 \end{bmatrix}$$

- Affine transformation – 4×4 matrix

$$\mathbf{M} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation

- Point $\mathbf{p}$ to $\mathbf{p'}$ by displacing by a distance d

$$\mathbf{p}' = \mathbf{p} + \mathbf{d} \longrightarrow \mathbf{p}' = T\mathbf{p}$$

$$x' = x + \alpha_x$$
$$y' = y + \alpha_y$$
$$z' = z + \alpha_z$$

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \quad \mathbf{p}' = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \\ 0 \end{bmatrix}$$

$$T = \begin{bmatrix} 1 & 0 & 0 & \alpha_x \\ 0 & 1 & 0 & \alpha_y \\ 0 & 0 & 1 & \alpha_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Inverse of a translation matrix

$$T^{-1}(\alpha_x, \alpha_y, \alpha_z) = T(-\alpha_x, -\alpha_y, -\alpha_z) = \begin{bmatrix} 1 & 0 & 0 & -\alpha_x \\ 0 & 1 & 0 & -\alpha_y \\ 0 & 0 & 1 & -\alpha_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling

- Scaling matrix with a fixed point of the origin

$$x' = \beta_x x$$
$$y' = \beta_y y$$
$$z' = \beta_z z$$

$$\xrightarrow{S(\beta_x, \beta_y, \beta_z)} \quad \mathbf{p}' = S\mathbf{p}$$

$$S = \begin{bmatrix} \beta_x & 0 & 0 & 0 \\ 0 & \beta_y & 0 & 0 \\ 0 & 0 & \beta_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Inverse of a scaling matrix

$$S^{-1}(\beta_x, \beta_y, \beta_z) = S\left(\frac{1}{\beta_x}, \frac{1}{\beta_y}, \frac{1}{\beta_z}\right) = \begin{bmatrix} 1/\beta_x & 0 & 0 & 0 \\ 0 & 1/\beta_y & 0 & 0 \\ 0 & 0 & 1/\beta_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation (1)

- Rotation with a fixed point at the origin

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
$$z' = z$$

$$\xrightarrow{R_z(\theta)} \quad \mathbf{p}' = R_z\mathbf{p}$$

$$R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_x = R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_y = R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation (2)

- Inverse of a rotation matrix

$$R^{-1}(\theta) = R(-\theta)$$

↓

$$\cos(-\theta) = \cos\theta, \quad \sin(-\theta) = -\sin\theta$$

↓

$$R_z^{-1}(\theta) = R_z(-\theta) = \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) & 0 & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

↓

$$\boxed{R^{-1} = R^T}$$: Orthogonal matrix

Concatenation of Transformations

- Concatenating
  - Affine transformations by multiplying together
  - Sequences of the basic transformations

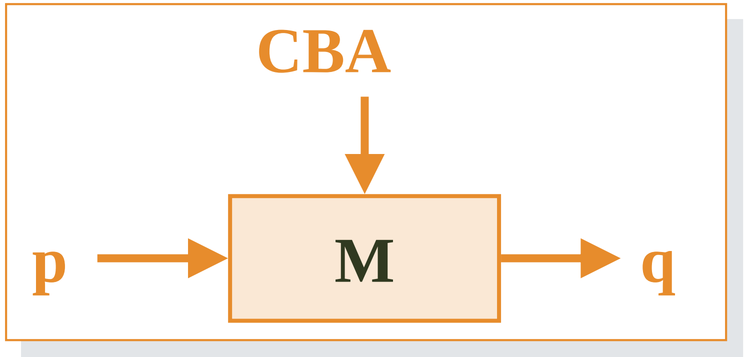
    ➔ Defining an arbitrary transformation directly
  - Ex) three successive transformations

$$\mathbf{p} \rightarrow \boxed{\mathbf{A}} \rightarrow \boxed{\mathbf{B}} \rightarrow \boxed{\mathbf{C}} \rightarrow \mathbf{q}$$

$$\mathbf{q} = (\mathbf{C}(\mathbf{B}(\mathbf{A}\mathbf{p}))) = \mathbf{CBAp}$$

$$\mathbf{M} = \mathbf{CBA}$$

$$\mathbf{q} = \mathbf{Mp}$$



Current Transformation Matrix

- Conceptually there is a 4x4 homogeneous coordinate matrix, the current transformation matrix (CTM) that is part of the state and is applied to all vertices that pass down the pipeline
- The CTM is defined in the user program and loaded into a transformation unit

CTM Operations

- The CTM can be altered either by loading a new CTM or by post-multiplication
  - Load an identity matrix: $\mathbf{C} \leftarrow \mathbf{I}$

  - Load a translation matrix: $\mathbf{C} \leftarrow \mathbf{T}$
  - Load a rotation matrix: $\mathbf{C} \leftarrow \mathbf{S}$
  - Load a scaling matrix: $\mathbf{C} \leftarrow \mathbf{R}$

  - Post-multiply by a translation matrix: $\mathbf{C} \leftarrow \mathbf{CT}$
  - Post-multiply by a rotation matrix: $\mathbf{C} \leftarrow \mathbf{CS}$
  - Post-multiply by a scaling matrix: $\mathbf{C} \leftarrow \mathbf{CR}$

Example: Rotation about a Fixed Point in WebGL

- Needs
  - Fixed point: (1, 2, 3)
  - Rotation angle: 30 degrees
  - Rotation axis : z axis

$$\mathbf{C} \leftarrow \mathbf{I}$$
$$\mathbf{C} \leftarrow \mathbf{CT}(1.0, 2.0, 3.0)$$
$$\mathbf{C} \leftarrow \mathbf{CR}(30.0, 0.0, 0.0, 1.0)$$
$$\mathbf{C} \leftarrow \mathbf{CT}(-1.0, -2.0, -3.0)$$

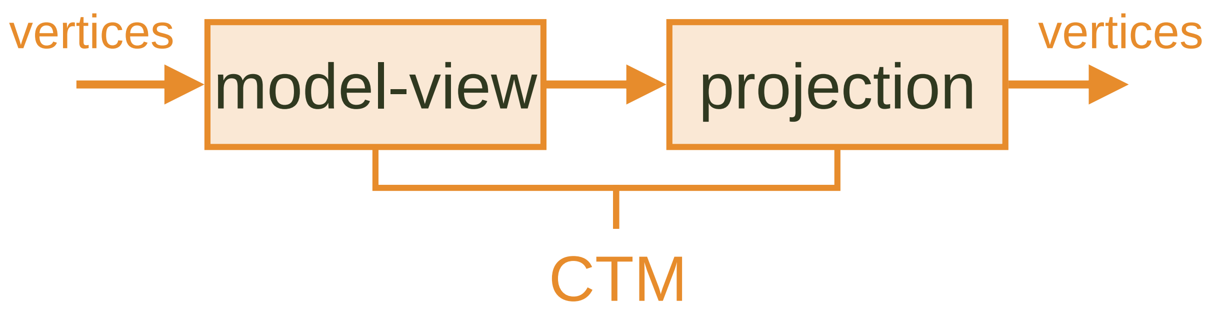
Example: Rotation about a Fixed Point in WebGL

- Needs
  - Fixed point: (1, 2, 3)
  - Rotation angle: 30 degrees
  - Rotation axis : z axis

```
mat4 m = Identity();
m = Translate( 1.0, 2.0, 3.0 ) *
    Rotate( 30.0, 0.0, 0.0, 1.0 ) *
    Translate( -1.0f, -2.0f, -3.0f );
```

CMT in WebGL

- OpenGL had a model-view and a projection matrix in the pipeline which were concatenated together to form the CTM



- We will emulate this process

Arbitrary Matrices

- Can load and multiply by matrices defined in the application program
- Matrices are stored as one dimensional array of 16 elements which are the components of the desired 4x4 matrix in row major order
- OpenGL wants column major data
- gl.uniformMatrix4f( ) has a parameter for automatic transpose by it must be set to false
- flatten( ) function converts to column major order which is required by WebGL functions

Matrix Stacks

- In many situations we want to save transformation matrices for use later
  - Traversing hierarchical data structures (Chapter 9)
- Pre 3.1 OpenGL maintained stacks for each type of matrix
- Easy to create same functionality in JS
  - push and pop are part of Array object

```
var stack = [];
stack.push(modelViewMatrix);
modelViewMatrix = stack.pop();
```

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Transformations</title>
        <script id="vertex-shader" type="x-shader/x-vertex">
        attribute vec4 vPosition;
        uniform float theta;

        void main() {
            float s = sin(theta);
            float c = cos(theta);
            gl_Position.x = c * vPosition.x - s * vPosition.y;
            gl_Position.y = s * vPosition.x + c * vPosition.y;
            gl_Position.z = 0.0;
            gl_Position.w = 1.0;
        }
        </script>

        <script id="fragment-shader" type="x-shader/x-fragment">
        precision mediump float;

        void main() {
            gl_FragColor = vec4(0.0, 1.0, 0.0, 1.0);
        }
        </script>

        <script type="text/javascript" src="Common/webgl-utils.js"></script>
        <script type="text/javascript" src="Common/initShaders.js"></script>
        <script type="text/javascript" src="Common/MV.js"></script>
        <script type="text/javascript" src="transform.js"></script>
```

```html
            float s = sin(theta);
            float c = cos(theta);
            gl_Position.x = c * vPosition.x - s * vPosition.y;
            gl_Position.y = s * vPosition.x + c * vPosition.y;
            gl_Position.z = 0.0;
            gl_Position.w = 1.0;
        }
        </script>

        <script id="fragment-shader" type="x-shader/x-fragment">
        precision mediump float;

        void main() {
            gl_FragColor = vec4(0.0, 1.0, 0.0, 1.0);
        }
        </script>

        <script type="text/javascript" src="Common/webgl-utils.js"></script>
        <script type="text/javascript" src="Common/initShaders.js"></script>
        <script type="text/javascript" src="Common/MV.js"></script>
        <script type="text/javascript" src="transform.js"></script>
    </head>
    <body>
        <canvas id="gl-canvas" width="512" height="512">
            Oops... your browser doesn't support the HTML5 canvas element!
        </canvas>
    </body>
</html>
```

```
var gl;
var theta = 0;
var thetaLoc;

window.onload = function init()
{
    var canvas = document.getElementById("gl-canvas");

    gl = WebGLUtils.setupWebGL(canvas);
    if( !gl ) {
        alert("WebGL isn't available!");
    }

    // Four vertices
    var vertices = [
        vec2(0, 0.5),
        vec2(-0.5, 0),
        vec2(0.5, 0),
        vec2(0, -0.5)
    ];

    // Configure WebGL
    gl.viewport(0, 0, canvas.width, canvas.height);
    gl.clearColor(0.9, 0.9, 0.9, 1.0);

    // Load shaders and initialize attribute buffers
    var program = initShaders(gl, "vertex-shader", "fragment-shader");
    gl.useProgram(program);

    // Load the data into the GPU
```

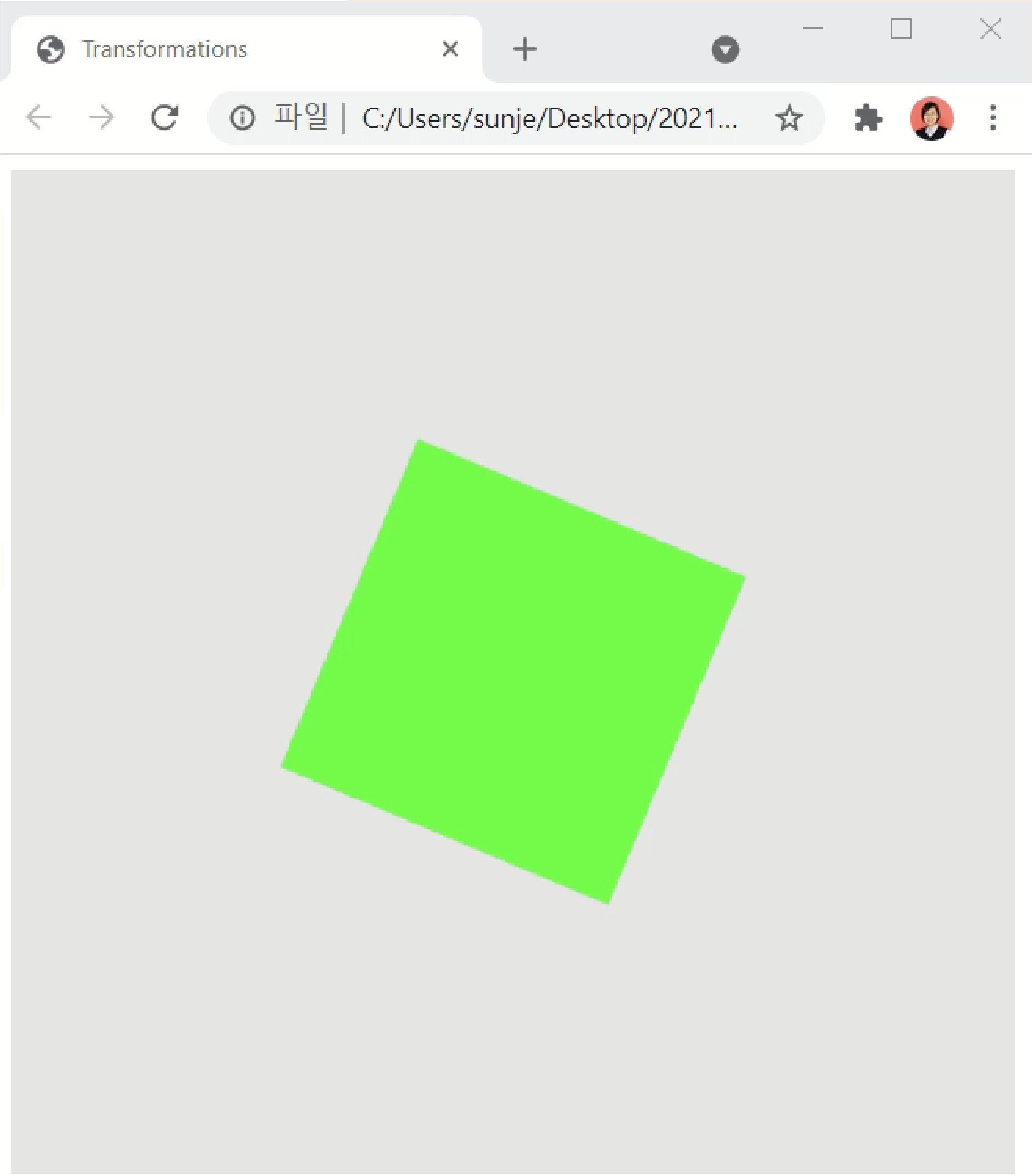
transform.js - Visual Studio Code

transform.html | transform.js

C: > Users > sunje > Desktop > 2021cg > transform.js > render

```javascript

    // Load the data into the GPU
    var bufferId = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, bufferId);
    gl.bufferData(gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW);

    // Associate our shader variables with our data buffer
    var vPosition = gl.getAttribLocation(program, "vPosition");
    gl.vertexAttribPointer(vPosition, 2, gl.FLOAT, false, 0, 0);
    gl.enableVertexAttribArray(vPosition);

    thetaLoc = gl.getUniformLocation(program, "theta");
    //gl.uniform1f(thetaLoc, theta);

    render();
};

function render() {
    gl.clear(gl.COLOR_BUFFER_BIT);

    theta += 0.1;
    gl.uniform1f(thetaLoc, theta);

    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);

    setTimeout(function() {
        window.requestAnimationFrame(render);
    }, 100);
}

```

Ln 56, Col 11   Spaces: 4   UTF-8   CRLF   JavaScript


Transformations
파일 | C:/Users/sunje/Desktop/2021...

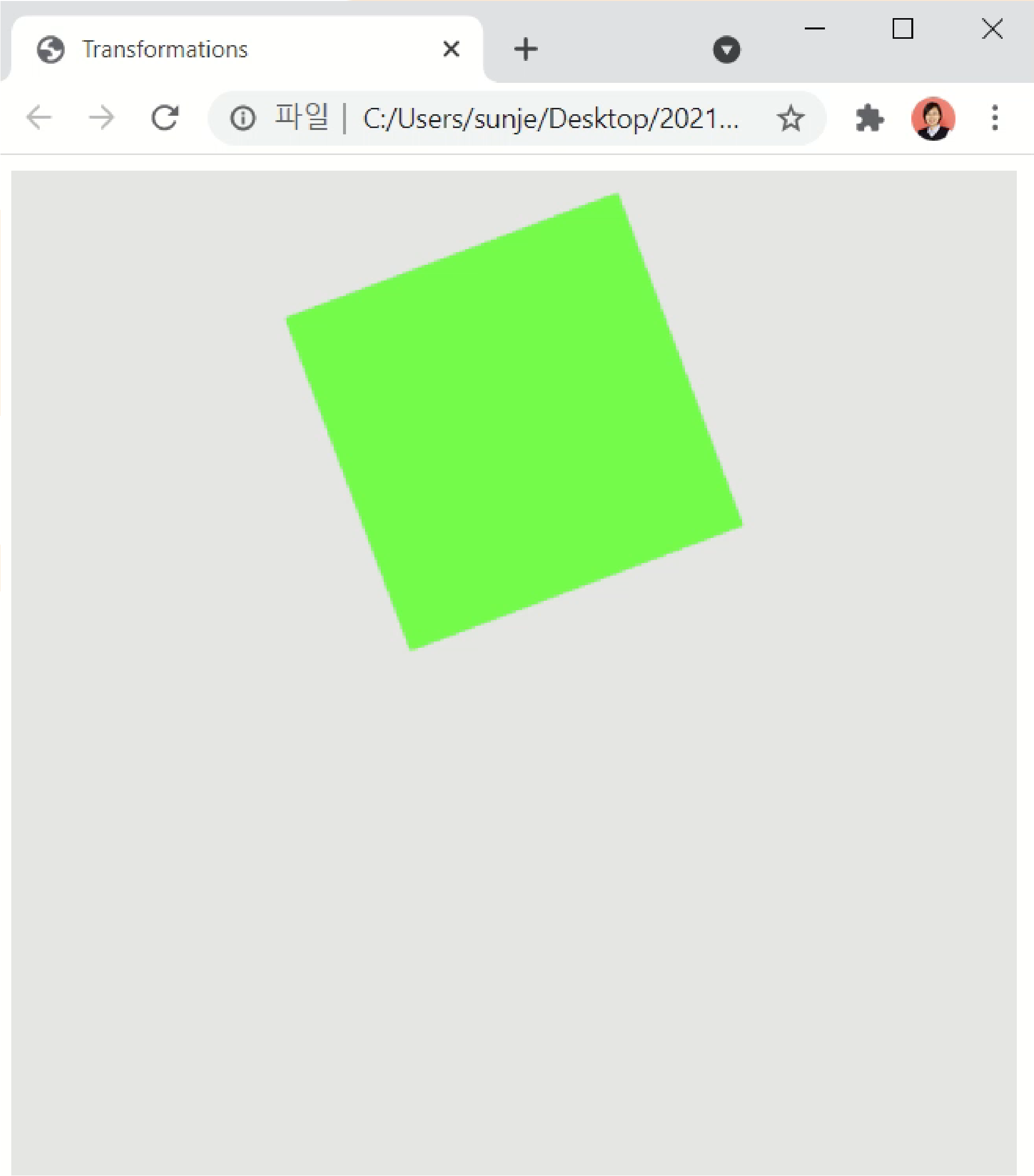
transform.html - Visual Studio Code

transform.html | transform.js

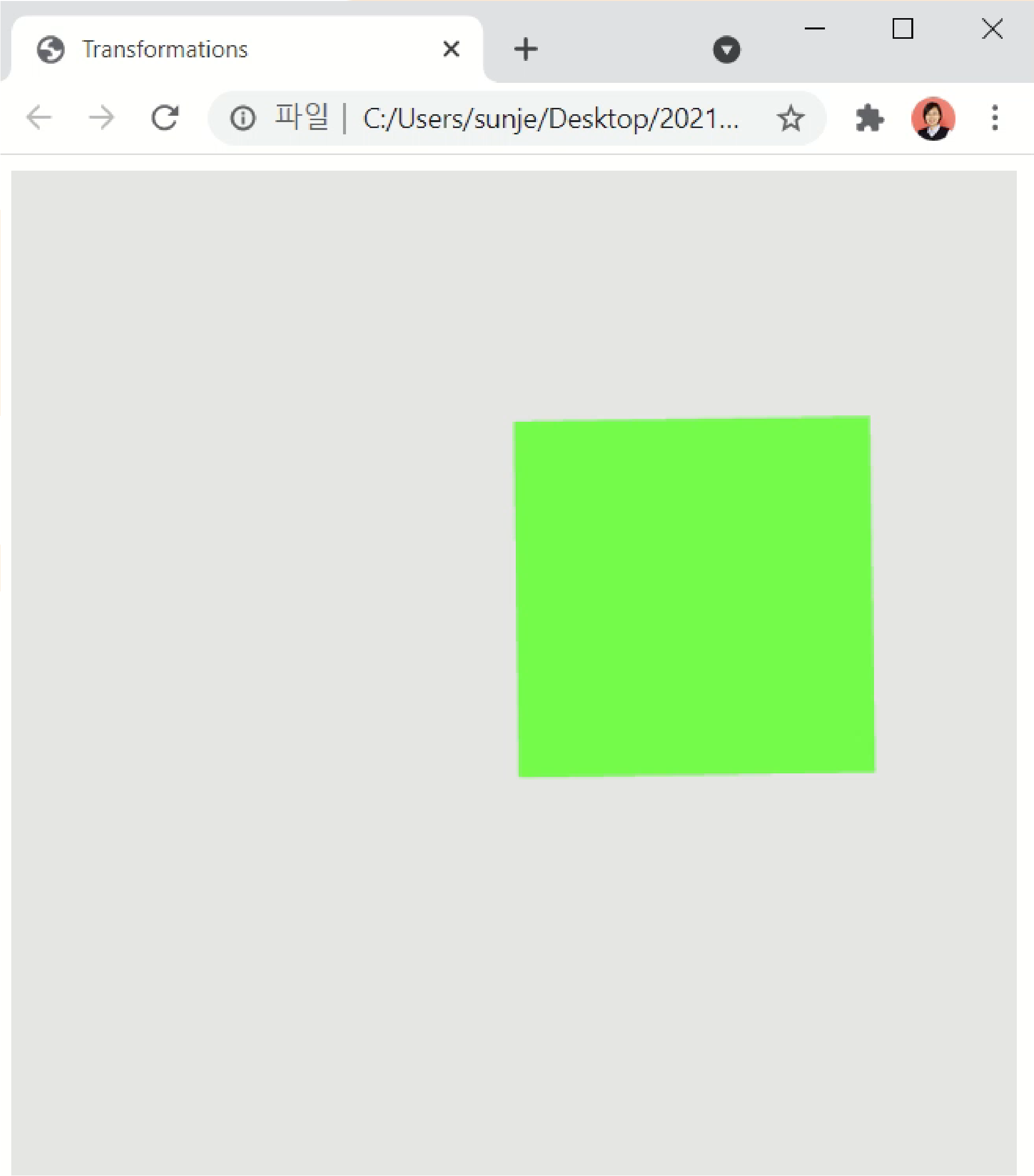
C: > Users > sunje > Desktop > 2021cg > transform.html > html > head > script#vertex-shader

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Transformations</title>
        <script id="vertex-shader" type="x-shader/x-vertex">
        attribute vec4 vPosition;
        uniform float theta;

        void main() {
            float s = sin(theta);
            float c = cos(theta);
            gl_Position.x = c * vPosition.x - s * vPosition.y;
            gl_Position.y = s * vPosition.x + c * vPosition.y + 0.5;
            gl_Position.z = 0.0;
            gl_Position.w = 1.0;
        }
        </script>

        <script id="fragment-shader" type="x-shader/x-fragment">
        precision mediump float;

        void main() {
            gl_FragColor = vec4(0.0, 1.0, 0.0, 1.0);
        }
        </script>

        <script type="text/javascript" src="Common/webgl-utils.js"></script>
        <script type="text/javascript" src="Common/initShaders.js"></script>
        <script type="text/javascript" src="Common/MV.js"></script>
        <script type="text/javascript" src="transform.js"></script>
```

Ln 13, Col 68 Spaces: 4 UTF-8 CRLF HTML

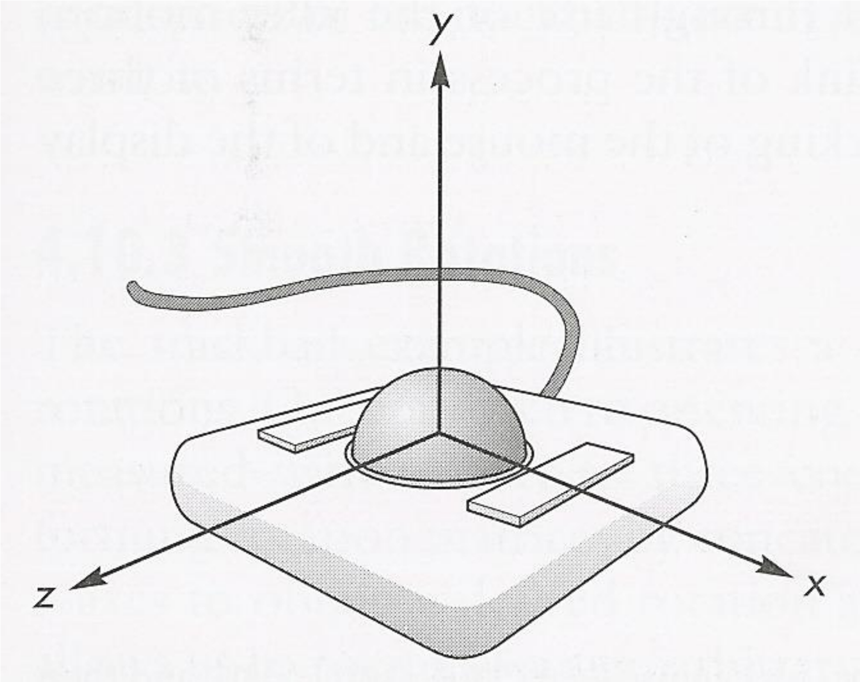

Transformations
파일 | C:/Users/sunje/Desktop/2021...

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Transformations</title>
        <script id="vertex-shader" type="x-shader/x-vertex">
        attribute vec4 vPosition;
        uniform float theta;

        void main() {
            float s = sin(theta);
            float c = cos(theta);
            gl_Position.x = c * vPosition.x - s * (vPosition.y-0.5);
            gl_Position.y = s * vPosition.x + c * (vPosition.y-0.5) + 0.5;
            gl_Position.z = 0.0;
            gl_Position.w = 1.0;
        }
        </script>

        <script id="fragment-shader" type="x-shader/x-fragment">
        precision mediump float;

        void main() {
            gl_FragColor = vec4(0.0, 1.0, 0.0, 1.0);
        }
        </script>

        <script type="text/javascript" src="Common/webgl-utils.js"></script>
        <script type="text/javascript" src="Common/initShaders.js"></script>
        <script type="text/javascript" src="Common/MV.js"></script>
        <script type="text/javascript" src="transform.js"></script>
```


Transformations
파일 | C:/Users/sunje/Desktop/2021...

Virtual Trackball (1)

- Using the mouse position to control rotation about two axes
- Supporting continuous rotations of objects



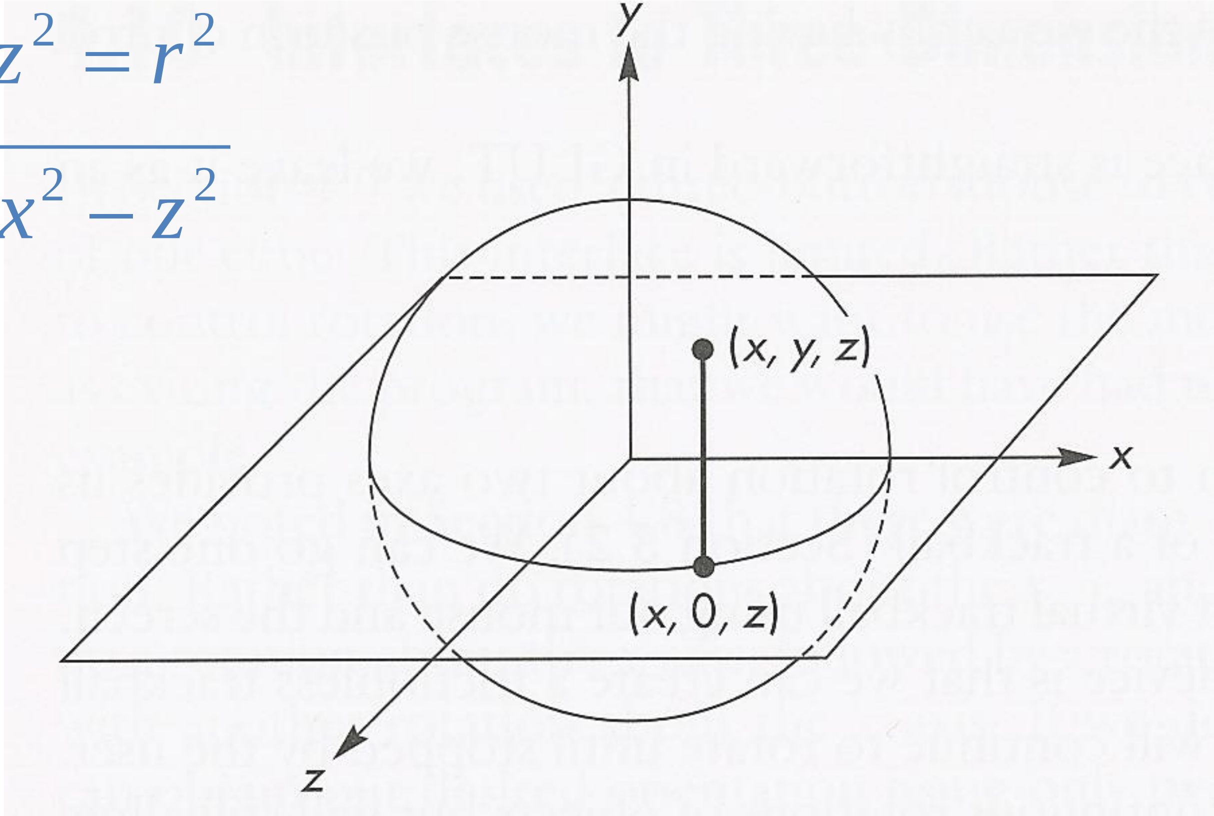
Trackball frame

Virtual Trackball (2)

- Rotation with a virtual trackball
  - Projection of the trackball position to the plane

$$x^2 + y^2 + z^2 = r^2$$

$$y = \sqrt{r^2 - x^2 - z^2}$$



Virtual Trackball (3)

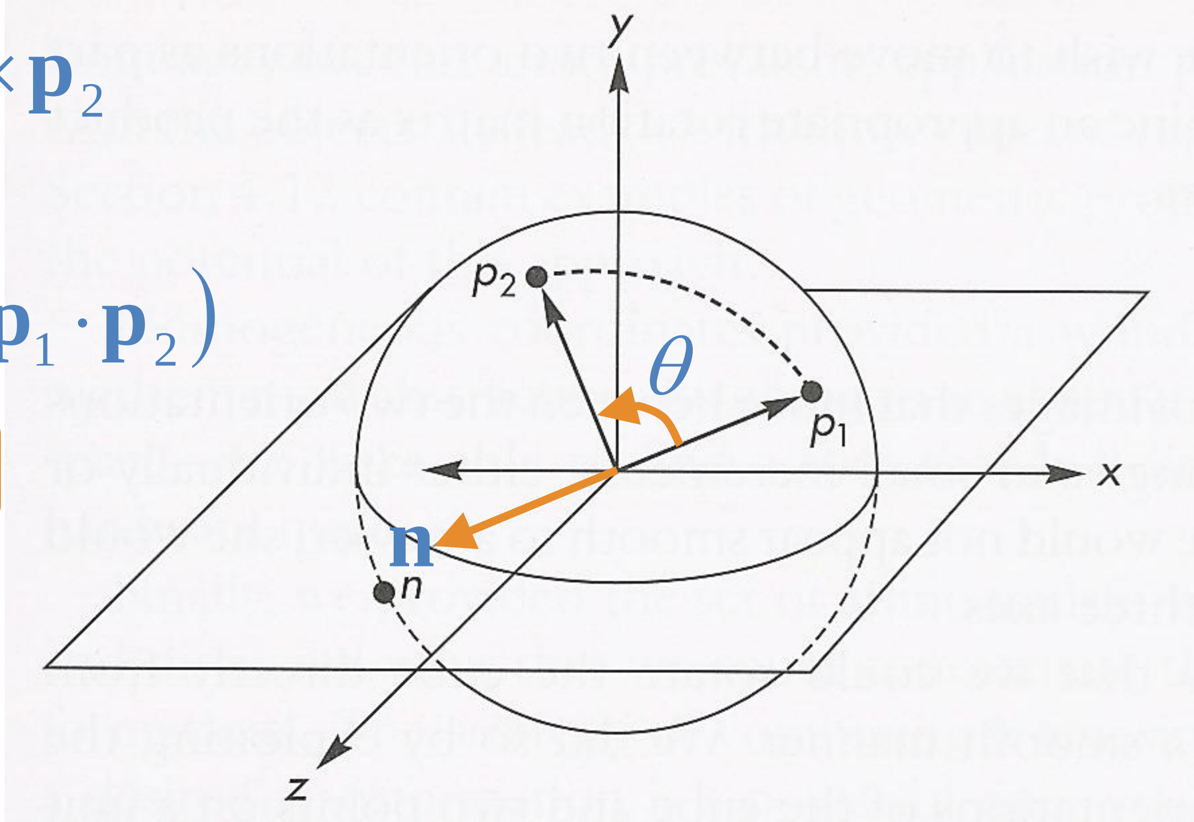
- Rotation with a virtual trackball (cont')
  - Determination of the orientation of a plane

$$\mathbf{n} = \mathbf{p}_1 \times \mathbf{p}_2$$

  - Rotation angle

$$\theta = \cos^{-1}(\mathbf{p}_1 \cdot \mathbf{p}_2)$$

➔ Quaternions



Complex Numbers (1)

- Real part + imaginary part: $z = x + iy$

$$\boxed{z = (x, y)}$$

$$x = \mathrm{Re}(z), \quad y = \mathrm{Im}(z)$$

- Addition and subtraction

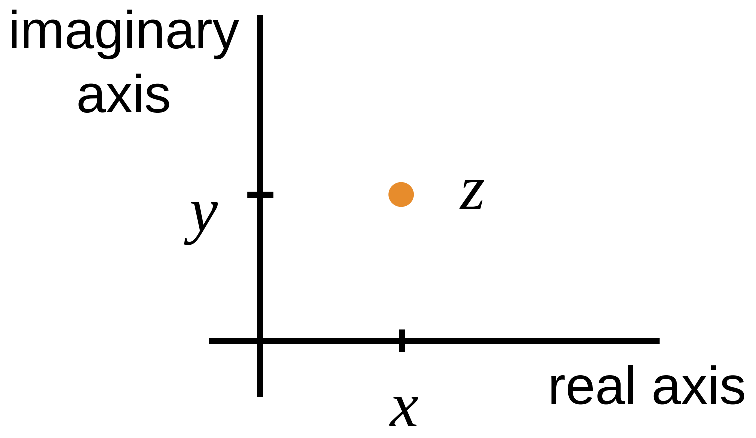
$$(x_1, y_1) \pm (x_2, y_2) = (x_1 \pm x_2, y_1 \pm y_2)$$

- Scalar multiplication

$$k(x_1, y_1) = (kx_1, ky_1)$$

- Multiplication

$$(x_1, y_1)(x_2, y_2) = (x_1x_2 - y_1y_2, x_1y_2 + x_2y_1)$$



Complex Numbers (2)

- Imaginary unit: $i = (0, 1)$

$$i^2 = (0,1)(0,1) = (-1, 0)$$

$$i = \sqrt{-1}$$

- Complex conjugate

$$z = x + iy \qquad \bar{z} = x - iy$$

  - Modulus or absolute value

$$|z| = z\bar{z} = \sqrt{x^2 + y^2}$$

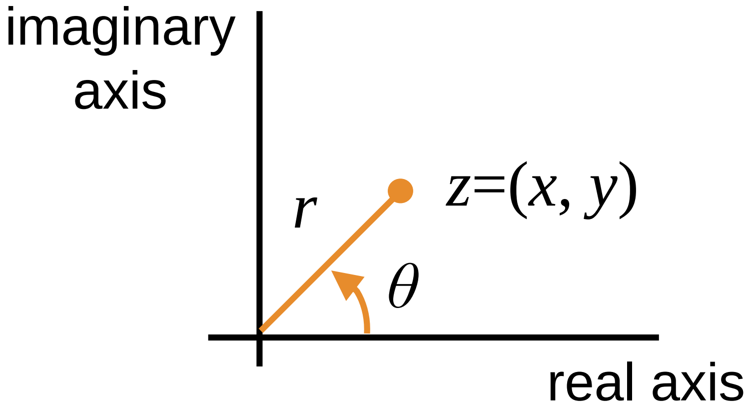
- Division

$$\frac{z_1}{z_2} = \frac{z_1\bar{z}_2}{z_2\bar{z}_2} = \frac{(x_1, y_1)(x_2, -y_2)}{x_2^2 + y_2^2} = \left(\frac{x_1x_2 + y_1y_2}{x_2^2 + y_2^2}, \frac{x_2y_1 - x_1y_2}{x_2^2 + y_2^2}\right)$$

Complex Numbers (3)

- Representation with polar coordinates

$$z = r(\cos\theta + i\sin\theta)$$



- Euler's formula

$$e^{i\theta} = \cos\theta + i\sin\theta$$

$$z = re^{i\theta}$$

- Complex multiplication and division

$$z_1 z_2 = r_1 r_2 e^{i(\theta_1+\theta_2)}, \quad \frac{z_1}{z_2} = \frac{r_1}{r_2} e^{i(\theta_1-\theta_2)}$$

- nth roots

$$\sqrt[n]{z} = \sqrt[n]{r}\left[\cos\left(\frac{\theta+2k\pi}{n}\right) + i\sin\left(\frac{\theta+2k\pi}{n}\right)\right], \quad k = 0, 1, 2, \cdots, n-1$$

Quaternions (1)

- One real part + three imaginary part $q = s + ia + jb + kc$
- Properties: $i^2 = j^2 = k^2 = -1$

$$ij = -ji = k$$
$$jk = -kj = i$$
$$ki = -ik = j$$

- Addition and scalar multiplication

$$q_1 + q_2 = (s_1 + s_2) + i(a_1 + a_2) + j(b_1 + b_2) + k(c_1 + c_2)$$
$$dq_1 = s_1 d + ia_1 d + jb_1 d + kc_1 d$$

Quaternions (2)

- Ordered-pair notation $$q = (s, \mathbf{v})$$
  - Scalar 's' + vector "$\mathbf{v} = (a, b, c)$"
- Addition: $$q_1 + q_2 = (s_1 + s_2, \mathbf{v}_1 + \mathbf{v}_2)$$
- Multiplication:
$$q_1 q_2 = (s_1 s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, s_1 \mathbf{v}_2 + s_2 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)$$
- Magnitude: $$|q|^2 = s^2 + \mathbf{v} \cdot \mathbf{v}$$
- Inverse: $$q^{-1} = \frac{1}{|q|^2}(s, \mathbf{v}) \Leftarrow qq^{-1} = q^{-1}q = (1, 0)$$

Quaternions and 3D Rotation

- For a 3D point (α, β, γ)
  - A unit quaternion $q=(s,a,b,c)$ and its conjugate $\bar{q}=(s,-a,-b,-c)$

$$q \cdot (0,\alpha,\beta,\gamma) \cdot \bar{q} = (0,\alpha',\beta',\gamma')$$

➔ ***Rotating (α, β, γ) by angle θ about the axis parallel to (u_x, u_y, u_z)***

- For $q=\left(\cos\frac{\theta}{2}, \sin\frac{\theta}{2}(u_x, u_y, u_z)\right)$ R_q is a 3D rotation about (u_x, u_y, u_z) by θ

$$R_q(p) = q \cdot p \cdot \bar{q}$$

Rotations with Quaternions (1)

- Rotation about an arbitrary axis
  - Setting up a unit quaternion ($\mathbf{u}$: unit vector)

$$s = \cos\frac{\theta}{2}, \ \ \mathbf{v} = \mathbf{u}\sin\frac{\theta}{2} = (a, b, c)$$

  - Representing any point position $\mathbf{P}$ in quaternion notation ($\mathbf{p} = (x, y, z)$)

$$\mathbf{P} = (0, \ \mathbf{p})$$

  - Carrying out with the quaternion operation ($q^{-1}=(s, -\mathbf{v})$)

$$\mathbf{P}' = q\mathbf{P}q^{-1}$$

  - Producing the new quaternion

$$\mathbf{P}' = (0, \ \mathbf{p}')$$

$$\mathbf{p}' = s^2\mathbf{p} + \mathbf{v}(\mathbf{p}\cdot\mathbf{v}) + 2s(\mathbf{v}\times\mathbf{p}) + \mathbf{v}\times(\mathbf{v}\times\mathbf{p})$$

Rotations with Quaternions (2)

- Obtaining the rotation matrix by quaternion multiplication

$$\mathbf{M}_R(\theta) = \begin{bmatrix} 1-2b^2-2c^2 & 2ab-2sc & 2ac+2sb \\ 2ab+2sc & 1-2a^2-2c^2 & 2bc-2sa \\ 2ac-2sb & 2bc+2sa & 1-2a^2-2b^2 \end{bmatrix}$$

$$= \mathbf{R}_x(-\theta_x)\mathbf{R}_y(-\theta_y)\mathbf{R}_z(\theta)\mathbf{R}_y(\theta_y)\mathbf{R}_x(\theta_x)$$

- Including the translations

$$\mathbf{R}(\theta) = \mathbf{T}^{-1}\mathbf{M}_R(\theta)\mathbf{T}$$

Example

- Rotation about z axis
  - Setting the unit quaternion: $s=\cos\frac{\theta}{2},\ \mathbf{v}=(0,\ 0,\ 1)\sin\frac{\theta}{2}$
  - Substituting $a=b=0$, $c=\sin(\theta/2)$ into the matrix:

$$\mathbf{M}_R(\theta)=\begin{bmatrix} 1-2\sin^2\frac{\theta}{2} & -2\cos\frac{\theta}{2}\sin\frac{\theta}{2} & 0 \\ 2\cos\frac{\theta}{2}\sin\frac{\theta}{2} & 1-2\sin^2\frac{\theta}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

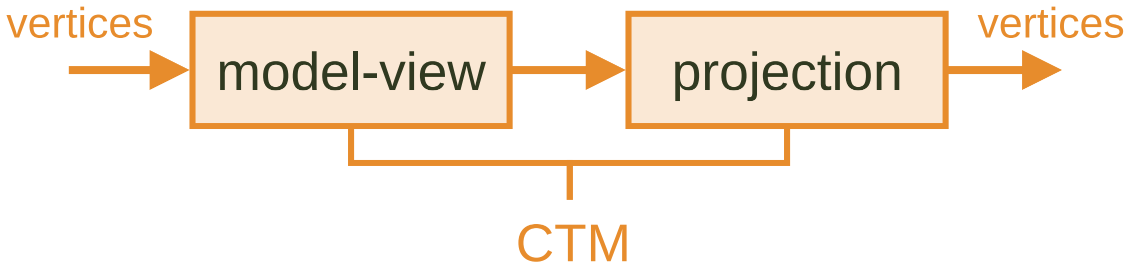
$$1-2\sin^2\frac{\theta}{2}=\cos\theta$$

$$2\cos\frac{\theta}{2}\sin\frac{\theta}{2}=\sin\theta$$

$$=\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Summary

- CTM (Current Transformation Matrix) in WebGL



  - The CTM can be altered either by loading a new CTM or by post-multiplication
    - Ex) $C = T^{-1}RT$
- Quaternions == rotation with a virtual trackball
  - Setting up a unit quaternion ($\mathbf{u}$: unit vector)

$$s = \cos\frac{\theta}{2}, \quad \mathbf{v} = \mathbf{u}\sin\frac{\theta}{2} = (a, b, c) \qquad \mathbf{M}_R(\theta) = \begin{bmatrix} 1-2b^2-2c^2 & 2ab-2sc & 2ac+2sb \\ 2ab+2sc & 1-2a^2-2c^2 & 2bc-2sa \\ 2ac-2sb & 2bc+2sa & 1-2a^2-2b^2 \end{bmatrix}$$

수고하셨습니다