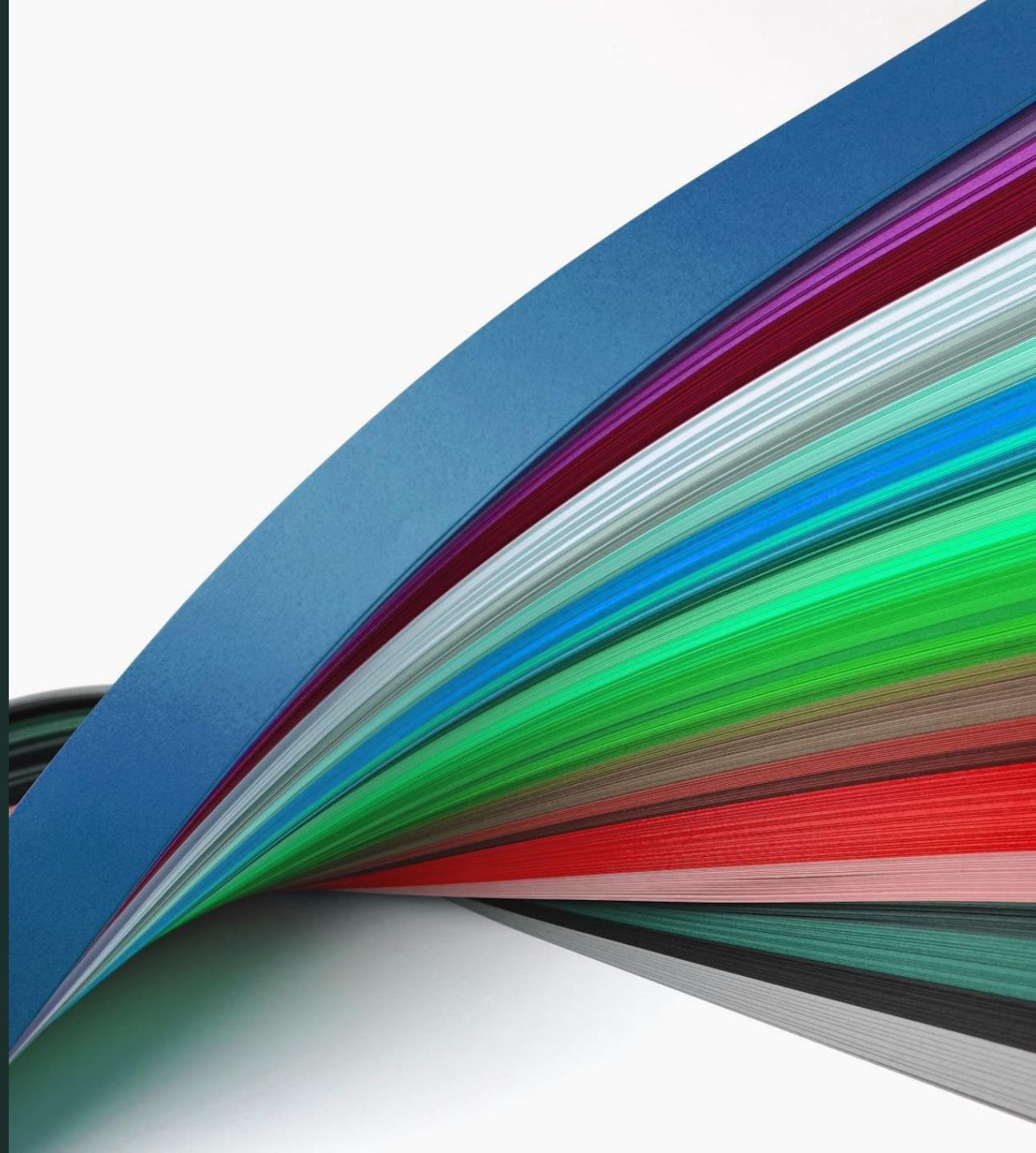


# XR-WWW1

Webudvikling



# Plan



1. HTML
2. CSS
3. Responsive Design
4. Individuel aflevering
5. JavaScript: Basics
6. JavaScript: JSON & Fetch
7. JavaScript: Sessions & Cookies
8. Gruppe aflevering

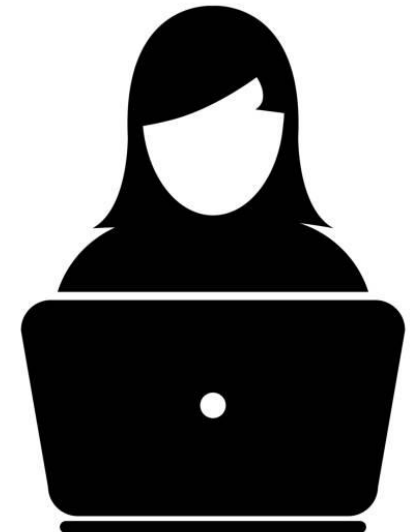
9. WebGL Simpel Trekant



## **10. WebGL Geometri, Tekstur & Rotation**

11. WebGL Belysning, Normalmaps

12. Afsluttende aflevering



# Opsummering



- Hvad er WebGL
- Hvordan virker WebGL
- InitWebGL()
- InitViewport()
- InitShaders()
- InitVertexShader() & InitFragmentShader()
- InitShaderProgram()
- ValidateShaderProgram()
- CreateGeometryBuffer()
- CreateVBO()
- Render()

## Praksis:

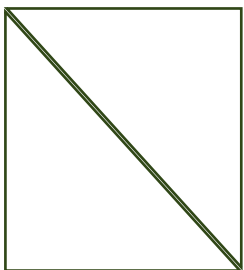
- Øvelse: Simpel Trekant

# WebGL



Hvordan tegner vi en firkant?

To sammensatte trekanter udgør en firkant.



Vi har hard-coded en trekant, lad os skrive en mere sofistikeret måde at lave trekanter på.

```
function CreateGeometryBuffers(program)
{
    // Triangle      X      Y      Z      R      G      B
    const vertices = [0.0, 0.5, 0.0, 1.0, 0.0, 0.0,
                     -0.5,-0.5, 0.0, 0.0, 1.0, 0.0,
                     0.5,-0.5, 0.0, 0.0, 0.0, 1.0];

    // Create GPU buffer (VBO)
    CreateVBO(program, new Float32Array(vertices));

    // Activate shader program
    gl.useProgram(program);

    // Display geometri on screen
    Render();
}
```



# WebGL



Brug 5 minutter til at tænke over, hvordan man kan undgå at hard-code trekanter i en array og i stedet lave en dynamisk måde at definere trekanter på.

Skriv dine løsninger ned og snak med den ved siden af dig og find ud af om I er kommet til samme løsning.

```
// Triangle      X      Y      Z      R      G      B
const vertices = [0.0, 0.5, 0.0, 1.0, 0.0, 0.0,
                  -0.5, -0.5, 0.0, 0.0, 1.0, 0.0,
                  0.5, -0.5, 0.0, 0.0, 0.0, 1.0];
```

Se gerne koden fra sidste slides i gennem for at finde en løsning.



# WebGL



Da Render() funktionen skal vide, hvor mange vertexer der skal renderes, bliver vi næsten nødt til at gemme trekanterne i en global array.

```
// Global variable
```

```
var vertices = [];
```

```
var gl = document.getElementById('gl')
```

```
.getContext('webgl') ||
```

```
// Support Internet Explorer, Edge, Safari
```

```
document.getElementById('gl')
```

```
.getContext('experimental-webgl');
```



# WebGL



Vi kan nu lave en funktion der hedder AddVertex() som vil tilføje en vertex til den globale array.

```
function AddVertex(x, y, z, r, g, b)
{
    const index = vertices.length;
    vertices.length += 6;
    vertices[index + 0] = x;
    vertices[index + 1] = y;
    vertices[index + 2] = z;
    vertices[index + 3] = r;
    vertices[index + 4] = g;
    vertices[index + 5] = b;
}
```



# WebGL



Dernæst kan vi lave en funktion, som vil tilføje en ny trekant til den globale array, ved at kalde `AddVertex()` tre gange.

```
function AddTriangle(x1, y1, z1, r1, g1, b1,  
                    x2, y2, z2, r2, g2, b2,  
                    x3, y3, z3, r3, g3, b3)  
{  
    AddVertex(x1, y1, z1, r1, g1, b1);  
    AddVertex(x2, y2, z2, r2, g2, b2);  
    AddVertex(x3, y3, z3, r3, g3, b3);  
}
```





# WebGL



Vi kan lave samme trekant som før, blot ved at kalde `AddTriangle()`.

```
function CreateGeometryBuffers(program)
{
    // Triangle X    Y    Z    R    G    B
    AddTriangle(0.0, 0.5, 0.0, 1.0, 0.0, 0.0,
               -0.5,-0.5, 0.0, 0.0, 1.0, 0.0,
               0.5,-0.5, 0.0, 0.0, 0.0, 1.0);

    // Create GPU buffer (VBO)
    CreateVBO(program, new Float32Array(vertices));

    // Activate shader program
    gl.useProgram(program);

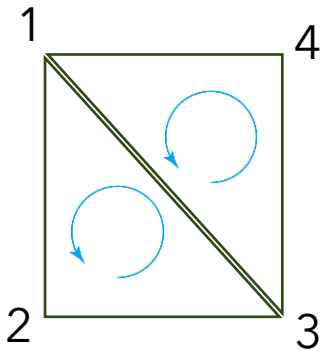
    // Display geometri on screen
    Render();
}
```



# WebGL



Det betyder at vi nu kan lave en firkant, blot ved at kalde `AddTriangle()` to gange.



```
function AddQuad(x1, y1, z1, r1, g1, b1,
                 x2, y2, z2, r2, g2, b2,
                 x3, y3, z3, r3, g3, b3,
                 x4, y4, z4, r4, g4, b4)
{
    AddTriangle(x1, y1, z1, r1, g1, b1,
               x2, y2, z2, r2, g2, b2,
               x3, y3, z3, r3, g3, b3);

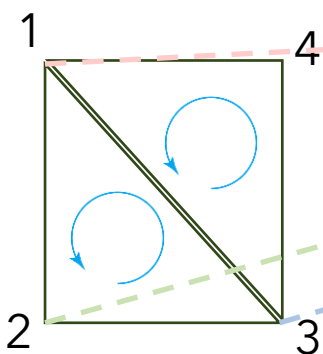
    AddTriangle(x3, y3, z3, r3, g3, b3,
               x4, y4, z4, r4, g4, b4,
               x1, y1, z1, r1, g1, b1);
}
```



# WebGL



Det betyder at vi nu kan lave en firkant, blot ved at kalde `AddTriangle()` to gange.



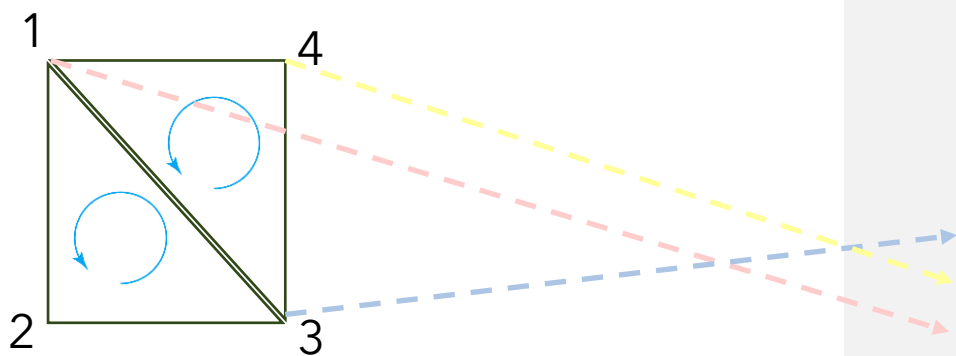
```
function AddQuad(x1, y1, z1, r1, g1, b1,  
                 x2, y2, z2, r2, g2, b2,  
                 x3, y3, z3, r3, g3, b3,  
                 x4, y4, z4, r4, g4, b4)  
{  
    AddTriangle(x1, y1, z1, r1, g1, b1,  
                x2, y2, z2, r2, g2, b2,  
                x3, y3, z3, r3, g3, b3);  
  
    AddTriangle(x3, y3, z3, r3, g3, b3,  
                x4, y4, z4, r4, g4, b4,  
                x1, y1, z1, r1, g1, b1);  
}
```



# WebGL



Det betyder at vi nu kan lave en firkant, blot ved at kalde AddTriangle() to gange.



```
function AddQuad(x1, y1, z1, r1, g1, b1,
                 x2, y2, z2, r2, g2, b2,
                 x3, y3, z3, r3, g3, b3,
                 x4, y4, z4, r4, g4, b4)
{
    AddTriangle(x1, y1, z1, r1, g1, b1,
                x2, y2, z2, r2, g2, b2,
                x3, y3, z3, r3, g3, b3);

    AddTriangle(x3, y3, z3, r3, g3, b3,
                x4, y4, z4, r4, g4, b4,
                x1, y1, z1, r1, g1, b1);
}
```



# WebGL



Vi kan nu let definere en firkant, blot ved at kalde `AddQuad()`.

```
function CreateGeometryBuffers(program)
{
    // Quad  X    Y    Z    R    G    B
    AddQuad(-0.5, 0.5, 0.0, 1.0, 0.0, 0.0,
           -0.5, -0.5, 0.0, 0.0, 1.0, 0.0,
           0.5, -0.5, 0.0, 0.0, 0.0, 1.0,
           0.5, 0.5, 0.0, 1.0, 1.0, 0.0);

    // Create GPU buffer (VBO)
    CreateVBO(program, new Float32Array(vertices));

    // Activate shader program
    gl.useProgram(program);

    // Display geometri on screen
    Render();
}
```



# WebGL



Husk at opdater Render() funktionen med antal af vertexer der skal renderes.

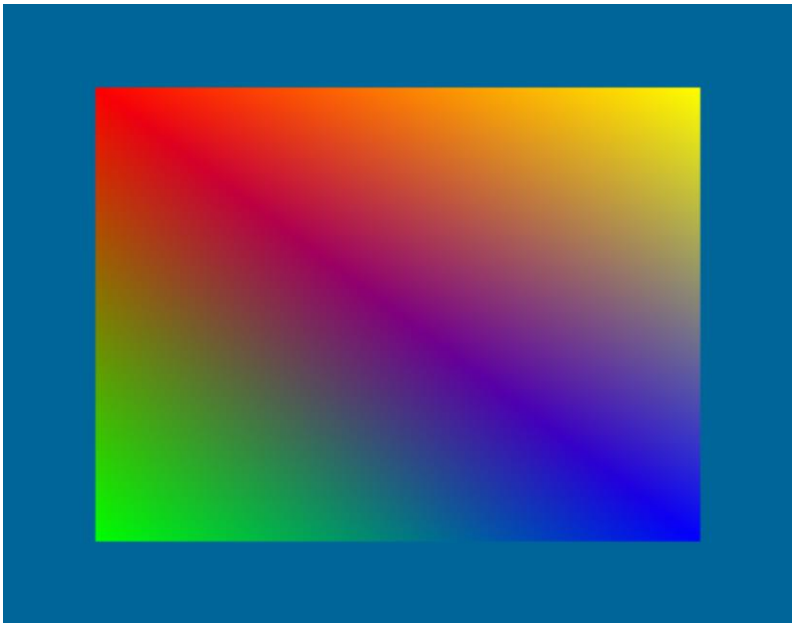
```
function Render()  
{  
    gl.clearColor(0.0, 0.4, 0.6, 1.0);  
    gl.clear(gl.COLOR_BUFFER_BIT |  
            gl.DEPTH_BUFFER_BIT );  
    gl.drawArrays(gl.TRIANGLES, 0, vertices.length / 6);  
}
```



# WebGL



Husk at opdater Render() funktionen med antal af vertexer der skal renderes.



```
function Render()  
{  
    gl.clearColor(0.0, 0.4, 0.6, 1.0);  
    gl.clear(gl.COLOR_BUFFER_BIT |  
            gl.DEPTH_BUFFER_BIT );  
    gl.drawArrays(gl.TRIANGLES, 0, vertices.length / 6);  
}
```



# WebGL



AddTriangle() tilføjer trekanter til et ønsket geometri.

Da man er tvunget til at definer, hver koordinat manuelt, virker den dog ikke særlig dynamisk.

Vi kan i stedet lave en anden funktion som generere koordinater baseret på dimensioner.

```
function CreateTriangle(width, height)
{
    vertices.length = 0;
    const w = width * 0.5;
    const h = height * 0.5;
    AddTriangle(0.0, h, 0.0, 1.0, 0.0, 0.0,
                -w, -h, 0.0, 0.0, 1.0, 0.0,
                w, -h, 0.0, 0.0, 0.0, 1.0);
}
```





# WebGL



AddQuad() tilføjer firkanter til et ønsket geometri.

Da man er tvunget til at definere, hver koordinat manuelt, virker den dog ikke særlig dynamisk.

Vi kan i stedet lave en anden funktion som generere koordinater baseret på dimensioner.

```
function CreateQuad(width, height)
{
    vertices.length = 0;
    const w = width * 0.5;
    const h = height * 0.5;
    AddQuad(-w, h, 0.0, 1.0, 0.0, 0.0,
            -w, -h, 0.0, 0.0, 1.0, 0.0,
            w, -h, 0.0, 0.0, 0.0, 1.0,
            w, h, 0.0, 1.0, 1.0, 0.0);
}
```



# WebGL



Lad os gøre det muligt for brugeren af websitet at vælge type af geometri der skal vises.

```
<body onload="InitWebGL();">
  <canvas id="gl" width="800px" height="600px">
    WebGL is not supported!
  </canvas>
  <textarea id="vs" spellcheck="false">
    <!-- Write vertex shader here! -->
  </textarea>
  <textarea id="fs" spellcheck="false">
    <!-- Write fragment shader here! -->
  </textarea>
  <select id="shape" onchange="InitShaders();">
    <option selected>Triangle</option>
    <option>Quad</option>
  </select>
  <script src="js/webgl.js" defer></script>
</body>
```



# WebGL



Vi kan nu afhængig af select elementet lave en geometrisk figur: Trekant eller Firkant.

Samt det eneste der er hard-coded er dimensionerne af geometrien.

Dette kunne laves 100% dynamisk, ved at tilføje HTML elementer, så som input number elementer, og lade JavaScript hente værdierne fra dem.

```
function CreateGeometryBuffers(program)
{
    let e = document.getElementById('shape');
    switch (e.selectedIndex)
    {
        case 0: CreateTriangle(1.0, 1.0); break;
        case 1: CreateQuad(1.0, 1.0); break;
    }
    // Create GPU buffer (VBO)
    CreateVBO(program, new Float32Array(vertices));
    // Activate shader program
    gl.useProgram(program);
    // Display geometri on screen
    Render();
}
```



# WebGL



Denne JavaScript funktion generere to input HTML elementer som hedder Width og Height under et elementet med id 'ui'.

```
function CreateGeometryUI() {  
    const ew = document.getElementById('w');  
    const w = ew ? ew.value : 1.0;  
    const eh = document.getElementById('h');  
    const h = eh ? eh.value : 1.0;  
    document.getElementById('ui').innerHTML =  
    'Width: <input type="number" id="w" value="' + w + '"  
    onchange= "InitShaders();"><br>' +  
    'Height: <input type="number" id="h" value="' + h + '"  
    onchange= "InitShaders();">';  
    let e = document.getElementById('shape');  
    switch (e.selectedIndex) {  
        case 0: CreateTriangle(1.0, 1.0); break;  
        case 1: CreateQuad(1.0, 1.0); break;  
    }  
}
```




# WebGL



Prøver at finde et HTML element med id: **w**

elementet med id 'ui'.



```
function CreateGeometryUI() {  
  const ew = document.getElementById('w');  
  const w = ew ? ew.value : 1.0;  
  const eh = document.getElementById('h');  
  const h = eh ? eh.value : 1.0;  
  document.getElementById('ui').innerHTML =  
    'Width: <input type="number" id="w" value="' + w + '"  
    onchange= "InitShaders();"><br>' +  
    'Height: <input type="number" id="h" value="' + h + '"  
    onchange= "InitShaders();">';  
  let e = document.getElementById('shape');  
  switch (e.selectedIndex) {  
    case 0: CreateTriangle(1.0, 1.0); break;  
    case 1: CreateQuad(1.0, 1.0); break;  
  }  
}
```

# WebGL



Denne JavaScript funktion generere to input HTML

Hvis HTML elementet findes, så initialiser  
**variablen w** med elementets value ellers sæt  
**variablen w** til 1.0

```
function CreateGeometryUI() {  
    const ew = document.getElementById('w');  
    const w = ew ? ew.value : 1.0;  
    const eh = document.getElementById('h');  
    const h = eh ? eh.value : 1.0;  
    document.getElementById('ui').innerHTML =  
        'Width: <input type="number" id="w" value="' + w + '"  
        onchange= "InitShaders();"><br>' +  
        'Height: <input type="number" id="h" value="' + h + '"  
        onchange= "InitShaders();">';  
    let e = document.getElementById('shape');  
    switch (e.selectedIndex) {  
        case 0: CreateTriangle(1.0, 1.0); break;  
        case 1: CreateQuad(1.0, 1.0); break;  
    }  
}
```



# WebGL



Denne JavaScript funktion generere to input HTML elementer som hedder Width og Height under et

Prøver at finde et HTML element med id: **h**



```
function CreateGeometryUI() {  
    const ew = document.getElementById('w');  
    const w = ew ? ew.value : 1.0;  
    const eh = document.getElementById('h');  
    const h = eh ? eh.value : 1.0;  
    document.getElementById('ui').innerHTML =  
        'Width: <input type="number" id="w" value="' + w + '"  
        onchange= "InitShaders();"><br>' +  
        'Height: <input type="number" id="h" value="' + h + '"  
        onchange= "InitShaders();">';  
    let e = document.getElementById('shape');  
    switch (e.selectedIndex) {  
        case 0: CreateTriangle(1.0, 1.0); break;  
        case 1: CreateQuad(1.0, 1.0); break;  
    }  
}
```

# WebGL



Denne JavaScript funktion generere to input HTML elementer som hedder Width og Height under et

Hvis HTML elementet findes, så initialiser **variablen h** med elementets value ellers sæt **variablen h** til 1.0



```
function CreateGeometryUI() {
  const ew = document.getElementById('w');
  const w = ew ? ew.value : 1.0;
  const eh = document.getElementById('h');
  const h = eh ? eh.value : 1.0;
  document.getElementById('ui').innerHTML =
    'Width: <input type="number" id="w" value="' + w + '"
    onchange= "InitShaders();"><br>' +
    'Height: <input type="number" id="h" value="' + h + '"
    onchange= "InitShaders();">';
  let e = document.getElementById('shape');
  switch (e.selectedIndex) {
    case 0: CreateTriangle(1.0, 1.0); break;
    case 1: CreateQuad(1.0, 1.0); break;
  }
}
```



# WebGL



Find HTML elementet med id: **ui** og indsæt følgende to HTML elementer:

Width: `<input type="number" id="w" value="1.0" onchange="InitShaders();">`

Height: `<input type="number" id="h" value="1.0" onchange="InitShaders();">`

```
function CreateGeometryUI() {  
    const ew = document.getElementById('w');  
    const w = ew ? ew.value : 1.0;  
    const eh = document.getElementById('h');  
    const h = eh ? eh.value : 1.0;  
    document.getElementById('ui').innerHTML =  
        'Width: <input type="number" id="w" value="' + w + '"  
        onchange= "InitShaders();"><br>' +  
        'Height: <input type="number" id="h" value="' + h + '"  
        onchange= "InitShaders();">';  
    let e = document.getElementById('shape');  
    switch (e.selectedIndex) {  
        case 0: CreateTriangle(1.0, 1.0); break;  
        case 1: CreateQuad(1.0, 1.0); break;  
    }  
}
```




# WebGL



Denne JavaScript funktion generere to input HTML elementer som hedder Width og Height under et elementet med id 'ui'.

Flyttet kode fra CreateGeometryBuffers()



```
function CreateGeometryUI() {  
    const ew = document.getElementById('w');  
    const w = ew ? ew.value : 1.0;  
    const eh = document.getElementById('h');  
    const h = eh ? eh.value : 1.0;  
    document.getElementById('ui').innerHTML =  
        'Width: <input type="number" id="w" value="' + w + '"  
        onchange= "InitShaders();"><br>' +  
        'Height: <input type="number" id="h" value="' + h + '"  
        onchange= "InitShaders();">';  
    let e = document.getElementById('shape');  
    switch (e.selectedIndex) {  
        case 0: CreateTriangle(1.0, 1.0); break;  
        case 1: CreateQuad(1.0, 1.0); break;  
    }  
}
```

# WebGL



Vi kan nu afhængig af select elementet lave en geometrisk figur: Trekant eller Firkant

Kald funktionen der vil generere geometri med tilhørende UI

dimensionerne af geometrien.

Dette kunne laves 100% dynamisk, ved at tilføje HTML elementer, så som input number elementer, og lade JavaScript hente værdierne fra dem.

```
function CreateGeometryBuffers(program)
{
    // Generate selected geometry and UI
    CreateGeometryUI();
    // Create GPU buffer (VBO)
    CreateVBO(program, new Float32Array(vertices));
    // Activate shader program
    gl.useProgram(program);
    // Display geometri on screen
    Render();
}
```




# WebGL



Tilføj et HTML **div** element med id: **ui**

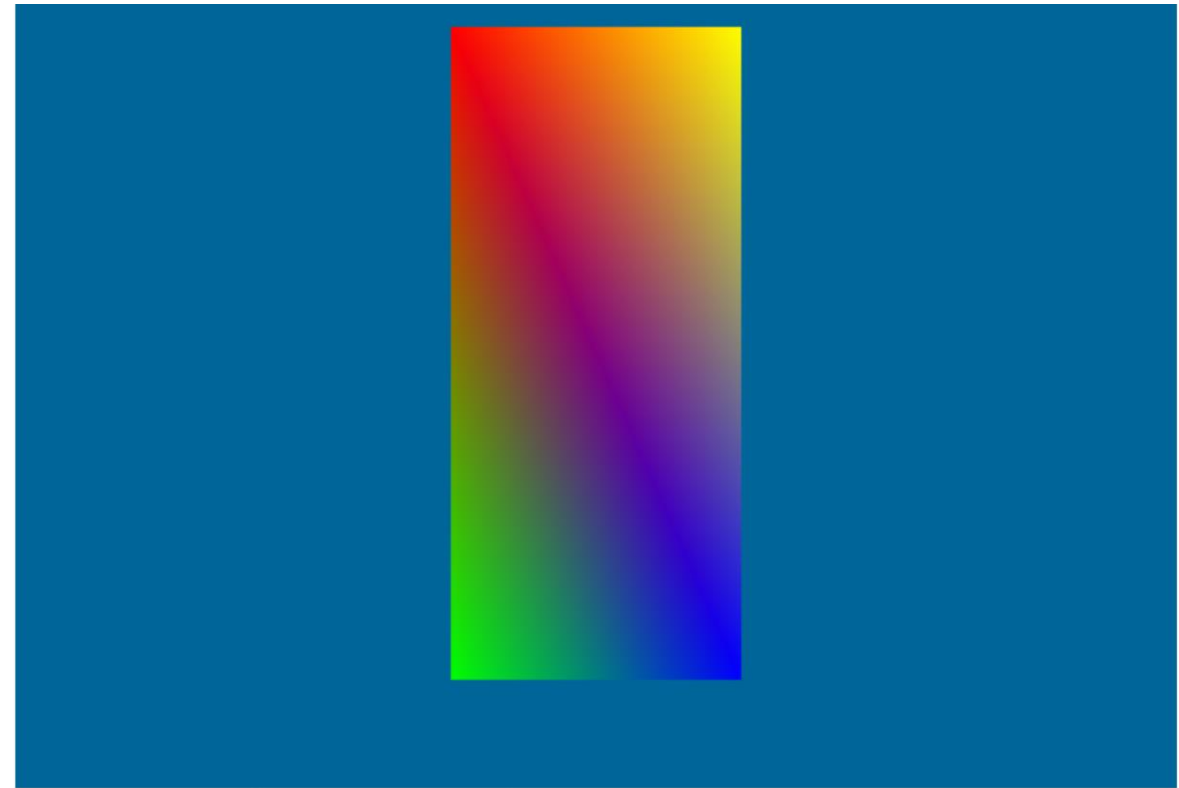
```
<body onload="InitWebGL();">
  <canvas id="gl" width="800px" height="600px">
    WebGL is not supported!
  </canvas>
  <textarea id="vs" spellcheck="false">
  </textarea>
  <textarea id="fs" spellcheck="false">
  </textarea>
  <select id="shape" onchange="InitShaders();">
    <option selected>Triangle</option>
    <option>Quad</option>
  </select>
  <div id="ui">Generate UI for geometry here!</div>
  <script src="js/webgl.js" defer></script>
</body>
```



# WebGL



Dit WebGL er nu 100% dynamisk, da bruger kan ændre dimensioner på det valgte geometri.



precision mediump float;	precision mediump float;
Quad	
Width: 0,5	
Height: 1,5	

# WebGL



Event Listeners i JavaScript giver dig mulighed for at oprette "begivenheder" (events), som vil eksekvere tilpasset funktioner på HTML elementer.

Her tilføjes en event listener, som opdaterer to globale variabler: **mouseX** og **mouseY**, når bruger bevæger musen over canvas elementet.

Andre ofte anvendte events er:  
click, mousedown, mouseup, mouseout, resize

En listener kan fjernes med `removeEventListener()`

```
var mouseX = 0, mouseY = 0;

document.getElementById('gl').addEventListener(
  'mousemove', function(e) {
    if (e.buttons == 1)
    {
      // Left mouse button pressed
    }
    mouseX = e.x;
    mouseY = e.y;
  });
```



# WebGL



Vi kan sende en array af floats, som en uniform vektor, til vores shader.

Denne array vil repræsentere rotations vinkler, der bliver genereret når musen bevæges.

```
var mouseX = 0, mouseY = 0;
var angle = [ 0.0, 0.0, 0.0, 1.0 ];
document.getElementById('gl').addEventListener(
  'mousemove', function(e) {
    if (e.buttons == 1)
    {
      // Left mouse button pressed
      angle[0] -= (mouseY - e.y) * 0.1;
      angle[1] += (mouseX - e.x) * 0.1;
    }
    mouseX = e.x;
    mouseY = e.y;
  });
```




# WebGL



**angleGL** er en uniform handler som kan bruges til at opdater grafikkortet med den ændrede array.

Når grafikkortet er opdateret, opdateres canvas elementet ved at kalde `Render()`, så vi kan se de nye ændringer.



```
var mouseX = 0, mouseY = 0;
var angle = [ 0.0, 0.0, 0.0, 1.0 ];
var angleGL = 0;
document.getElementById('gl').addEventListener(
  'mousemove', function(e) {
    if (e.buttons == 1)
    {
      // Left mouse button pressed
      angle[0] -= (mouseY - e.y) * 0.1;
      angle[1] += (mouseX - e.x) * 0.1;
      gl.uniform4fv(angleGL, new Float32Array(angle));
      Render();
    }
    mouseX = e.x;
    mouseY = e.y;
  });
```



# WebGL



Husk at fortælle grafikortet om at den skal lave en uniform location i shaderen med navnet: **Angle**

```
function CreateGeometryBuffers(program)
{
    // Generate selected geometry and UI
    CreateGeometryUI();

    // Create GPU buffer (VBO)
    CreateVBO(program, new Float32Array(vertices));

    angleGL = gl.getUniformLocation(program, 'Angle');

    // Activate shader program
    gl.useProgram(program);

    // Display geometri on screen
    Render();
}
```



# WebGL



Vi kan nu tilføje uniform vektor: **Angle** til vertex shaderen og sætte to matrixer op som roterer geometrien omkring sig selv.

```
precision mediump float;
attribute vec3 Pos;
attribute vec3 Color;
uniform vec4 Angle;
varying vec3 vertexColor;
void main() {
    float coX = cos(angle.x);
    float siX = sin(angle.x);
    mat4 matX = mat4(vec4(1.0, 0.0, 0.0, 0.0),
                    vec4(0.0, coX, siX, 0.0),
                    vec4(0.0, -siX, coX, 0.0),
                    vec4(0.0, 0.0, 0.0, 1.0));

    float coY = cos(angle.y);
    float siY = sin(angle.y);
    mat4 matY = mat4(vec4(coY, 0.0, -siY, 0.0),
                    vec4(0.0, 1.0, 0.0, 0.0),
                    vec4(siY, 0.0, coY, 0.0),
                    vec4(0.0, 0.0, 0.0, 1.0));

    gl_Position = matY * matX * vec4(Pos, 1.0);
    vertexColor = Color;
}
```

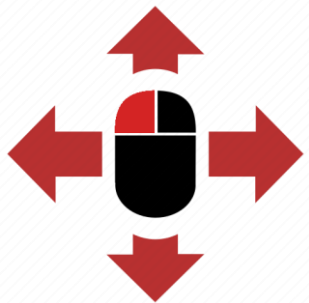


# WebGL



Vi kan nu tilføje uniform vektor: **Angle** til vertex shaderen og sætte to matrixer op som roterer geometrien omkring sig selv.

Bruger vil nu kunne rotere geometrien, ved at bevæge musen mens venstre museknap holdes nede!



```
precision mediump float;
attribute vec3 Pos;
attribute vec3 Color;
uniform vec4 Angle;
varying vec3 vertexColor;
void main() {
    float coX = cos(angle.x);
    float siX = sin(angle.x);
    mat4 matX = mat4(vec4(1.0, 0.0, 0.0, 0.0),
                    vec4(0.0, coX, siX, 0.0),
                    vec4(0.0, -siX, coX, 0.0),
                    vec4(0.0, 0.0, 0.0, 1.0));

    float coY = cos(angle.y);
    float siY = sin(angle.y);
    mat4 matY = mat4(vec4(coY, 0.0, -siY, 0.0),
                    vec4(0.0, 1.0, 0.0, 0.0),
                    vec4(siY, 0.0, coY, 0.0),
                    vec4(0.0, 0.0, 0.0, 1.0));

    gl_Position = matY * matX * vec4(Pos, 1.0);
    vertexColor = Color;
}
```



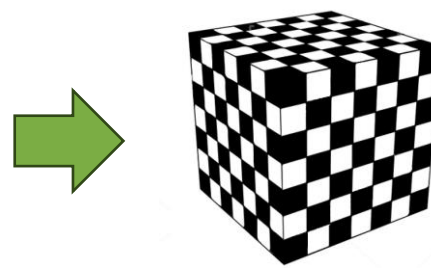
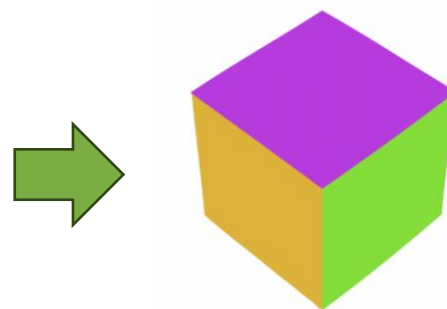
# WebGL



Vi venter med tekstur næste gang og fokuser i stedet på at generere geometri

Øvelse: **Geometri, ~~Tekstur~~ & Rotation**

- 1) Implementer alt kode som er vist i dette oplæg.
- 2) Tilføj en funktion som kan generere en 3D kasse ved blot at bruge AddQuad(). Lad siderne have forskellige farver, så det er let at se, hvilke side der er hvad.
- 3) Gør det muligt for bruger at 'subdivide' kassen. Generer UI til opdeling og farv quads sort/hvid.



Width	<input type="text" value="1.0"/>	Divs X	<input type="text" value="6"/>
Height	<input type="text" value="1.0"/>	Divs Y	<input type="text" value="6"/>
Depth	<input type="text" value="1.0"/>	Divs Z	<input type="text" value="6"/>

