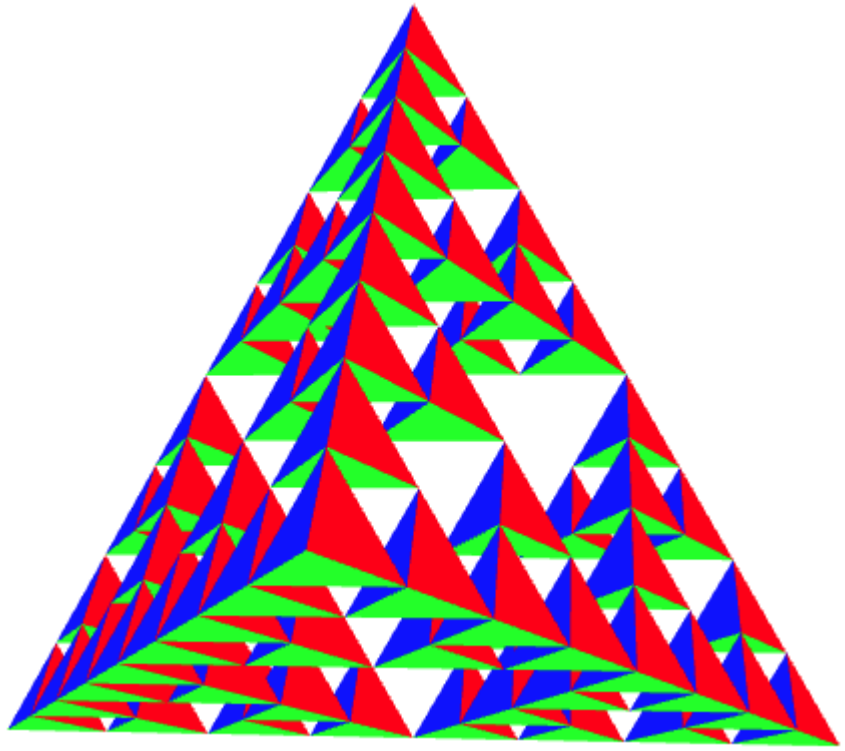
TÖL105M TÖLVUGRAFÍK

Fyrirlestur 5: Gagnvirkni



Hjálmtýr Hafsteinsson
Haust 2024

Í þessum fyrirlestri

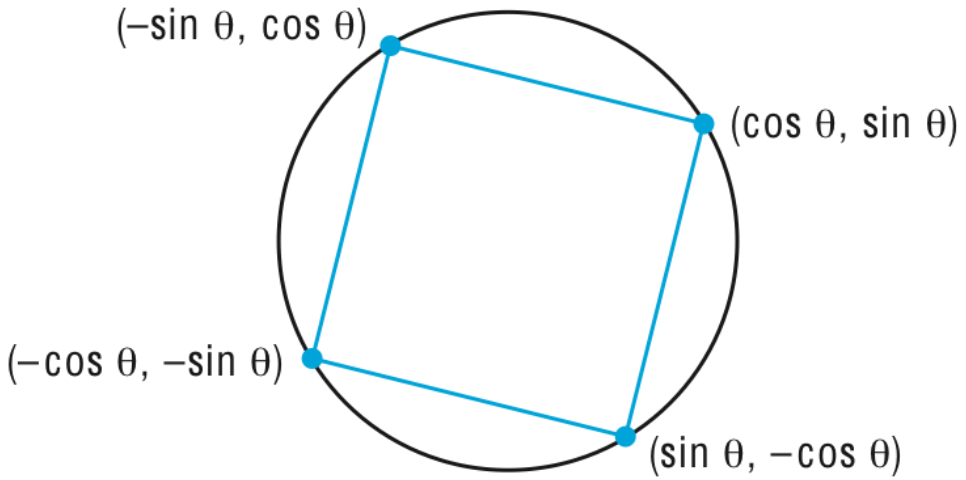
- Hreyfimyndir
  - Breyta hnitum hnúta
  - Tvö sýniforrit
- Atburðadrifið inntak
  - Músaatburðir
  - Lyklaborðsatburðir

3.1 – 3.2

3.6 – 3.11

Hreyfimyndir (*animation*)

- Ferningur hefur fjóra hnúta (*vertices*)
  - Snúum ferningnum með því að snúa hnútunum um θ gráður um miðju ferningsins
  - Hreyfingin fæst með því að hreinsa skjáinn og endurbirta ferninginn með örlítið breyttu gildi á θ



Snúningur (*rotation*)

- Snúa punktinum (x, y) rangsælis um hornið θ
  - Táknum punktinn í pólhnitum:

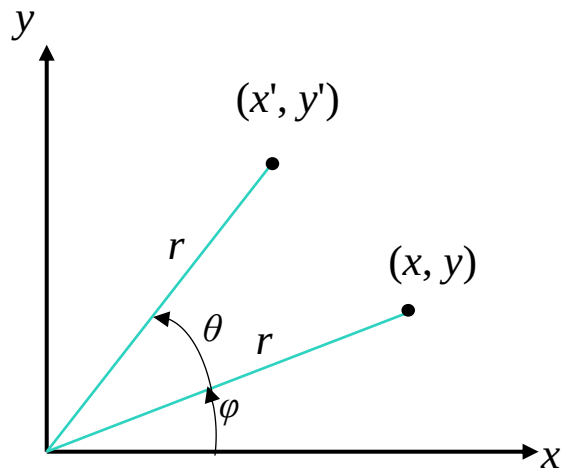
$$x = r\cos\phi$$
$$y = r\sin\phi$$

umritum:

$$x' = r\cos(\phi+\theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta$$
$$y' = r\sin(\phi+\theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta$$

setjum inn fyrir *x* og *y*:

$$\boxed{\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \end{aligned}}$$



Tvær útgáfur

- Látum JS forrit breyta hnitum hnútanna og senda breyttu hnitin yfir til GPU
  - Sýniforrit **`rotatingSquare0`** (ekki á heimasíðu bókar)

- Láta JS forrit hækka snúningshorn (θ), senda það yfir til GPU og láta litara breyta hnitum hnútanna
  - Sýniforrit **`rotatingSquare1`** (á heimasíðu bókar)

`rotatingSquare0` (reikna ný hnit í JS forriti og senda)

- Hnúta- og bútalitara eru einfaldir (*pass-through*):

```
attribute vec4 vPosition;

void main() {
   gl_Position = vPosition;
}
```

```
precision mediump float;

void main() {
   gl_FragColor = vec4( 1.0, 0.0, 0.0, 1.0 );
}
```

- Í JS forriti: skilgreinum upphafleg hnit hnúta og sendum þau yfir:

```
var v = [vec2(0, 1), vec2(-1, 0), vec2(1, 0), vec2(0, -1)];
...
var bufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
gl.bufferData( gl.ARRAY_BUFFER, flatten(v), gl.STATIC_DRAW );
```

`rotatingSquare0` (reikna ný hnit í JS forriti og senda)

- Í **`render`**-falli:

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

    var s = Math.sin(theta);
    var c = Math.cos(theta);
    for (var i=0; i<4; i++) {
       v[i] = vec2(c*v[i][0] - s*v[i][1], s*v[i][0] + c*v[i][1]);
    }

    gl.bufferSubData(gl.ARRAY_BUFFER, 0, flatten(v));

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

    window.requestAnimFrame(render);
}
```

Hreinsa skjáinn

Forreiknum sin θ og cos θ

Breyta hnitum allra hnútanna samkvæmt sin θ og cos θ

Ath: `v[i][0]` er *x*-hnit hnútar `i` og `v[i][1]` er *y*-hnit hans

Senda breyttu hnitin til GPU

Biðja vafra um endurteikningu með fallinu **`render`**

Eiginleikar `rotatingSquare0`

- Höfum hnit allra hnúta í JS forriti
  - Getum skoðað gildin og aflúsað forritið

- CPU að vinna óþarfa vinnu
  - JS er frekar hægvirkt forritunarmál
- Sendum "mikið" gagnamagn til GPU
  - Í hverri ítrun eru ný hnit allra hnúta send

`rotatingSquare1` (senda aðeins snúningshorn yfir í GPU)

- Í JS forriti: finna staðsetningu **`uniform`**-breytu í litara:

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

- Í **`render`**-falli: hækka snúningshorn og senda yfir:

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

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

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

    window.requestAnimFrame(render);
}
```

Aðeins einu gildi breytt

Senda nýja gildið yfir til GPU

`rotatingSquare1` (senda aðeins snúningshorn yfir í GPU)

- Hnútalitarinn breytir upphaflegum hnitum samkvæmt **`uniform`** breytunni **`theta`**:

```
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;
}
```

Forreiknum sinθ og cos θ

Breyta hnitum núverandi hnútar samkvæmt því

Hversu oft reiknar GPU-ið cos θ (og sin θ)?

Væri kannski betra að senda frekar cos θ og sin θ?

Eiginleikar `rotatingSquare1`

- Hnitum hnútanna breytt í litara
  - Erfitt að sjá og aflúsa gildi þeirra

- GPU vinnur mestu vinnuna
  - Hannað fyrir slíka vinnslu, hraðvirkt
- Mjög lítið gagnamagn sent
  - Í hverri ítrun er aðeins eitt `float` gildi sent

Samanburður

- Í `rS0` er hnitum hnútanna snúið um 0.1 gráðu í hverri ítrun
  - Hnit hnútanna breytast!

- Í `rS1` er snúningshornið θ hækkað um 0.1 gráðu í hverri ítrun og litarinn snýr upphaflegu hnitum hnútanna um það horn
  - Hnit hnútanna í grafíkminni breytast aldrei!
  - Breytast bara á meðan þau eru í grafíkpípunni

Ath: Í `rS1` eru `sin(theta)` og `cos(theta)` reiknuð fyrir hvern einasta hnút. Er til betri leið?

Fyrirlestraæfingar

1. Í forritinu **`rotatingSquare1`** er breytan **`theta`** hækkuð í hverri ítrun. Hvaða áhætta er fólgin í því?
2. Sýnið atburðafall fyrir "**`mousedown`**", þannig að vinstri músarsmellur hækkar hraðabreytuna **`speed`** um 10%, en hægri smellur lækkar hana um 10%.
3. Ef striginn (*canvas*) er 100x100 og við smellum á skjápunkt (25,75), hvaða gildi hefur sá punktur í heimshnitum?

Algeng aðferðafræði í WebGL

- Senda upphafleg gögn yfir til GPU í upphafi
  - Hnútahnit hluta, t.d. fernings, tenings

- Í **`render`**-falli:
  - Breyta einhverjum eiginleika (tíma, gráður, ...)
  - Senda nýtt gildi yfir til GPU
  - Teikna aftur

Þetta gildi er oftast uppsöfnuð breyting, ekki bara Δt

Litaraforrit breytir upphaflegum gögnum í samræmi við nýtt gildi og birtir nýja mynd

Uppfærsluhraði

- Ef við notum **`requestAnimFrame`**, þá er reynt að kalla á **`render`**-fallið 60 sinnum á sek.
  - Fáum þá "mjúka" hreyfingu
  - En erfitt að stýra hraða forritsins

- Javascript hefur tvö föll til að stýra uppfærsluhraða:
  - **`setInterval(render, 100)`**
  - **`setTimeout(render, 100)`**

Keyra **`render`** á 100ms fresti

Gallar: Óháð vafra. Getur "neytt" vafra til að uppfæra áður en hann tilbúinn

Betri útgáfa

- Blanda saman **`requestAnimFrame`** og **`setTimeout`**
  - Látum fallið í **`setTimeout`** byrja á því að kalla á **`requestAnimFrame`**
    - Þá fær vafrinn tækifæri til að klára önnur verk áður en grafík uppfærð

```
function render() {
    setTimeout( function() {
        window.requestAnimFrame(render);
        gl.clear( gl.COLOR_BUFFER_BIT );
        theta += 0.1;
        gl.uniform1f( thetaLoc, theta );
        gl.drawArrays( gl.TRIANGLE_STRIP, 0, 4 );
    }, 100);
}
```

Sjá **`rotatingSquare1time`**

Inntak í tölvugrafík

- Grafísk forritasöfn leiða inntakstæki hjá sér
  - Tæknin breytist og betra að nota rökrænt inntak
- Inntaksgerðir:
  - Atburðadrifið (*event driven*)
    - Músarsmellur og lyklaborðsásláttur
  - Beiðnadrifið (*request driven*)
    - Slá inn texta og ýta á **Enter**

Flest grafíkforrit eru atburðadrifin
Skilgreinum viðbrögð við atburðunum

Javascript atburðir (*events*)

- Músaatburðir (algengustu):
  - **`mousedown`**
  - **`mouseup`**
  - **`mousemove`**
- Lyklaborðsáslættur:
  - **`keydown`**
  - **`keyup`**

Fáum músarsmell

Fylgjast með hreyfingu músar

Mest notaður

Meðhöndlun atburða

- Hver atburður hefur "*target*", hlut sem atburðurinn verður á (hnappur, strigi, ...)
- Skráum föll sem bregðast við tilteknum atburðum (*event handlers*)

- Sýnidæmi: **`rotatingSquare1click`**
  - Ef vinstri-músarsmellur á striganum þá stoppar eða byrjar snúningur
  - Ef hægri-músarsmellur þá snýst snúningurinn við

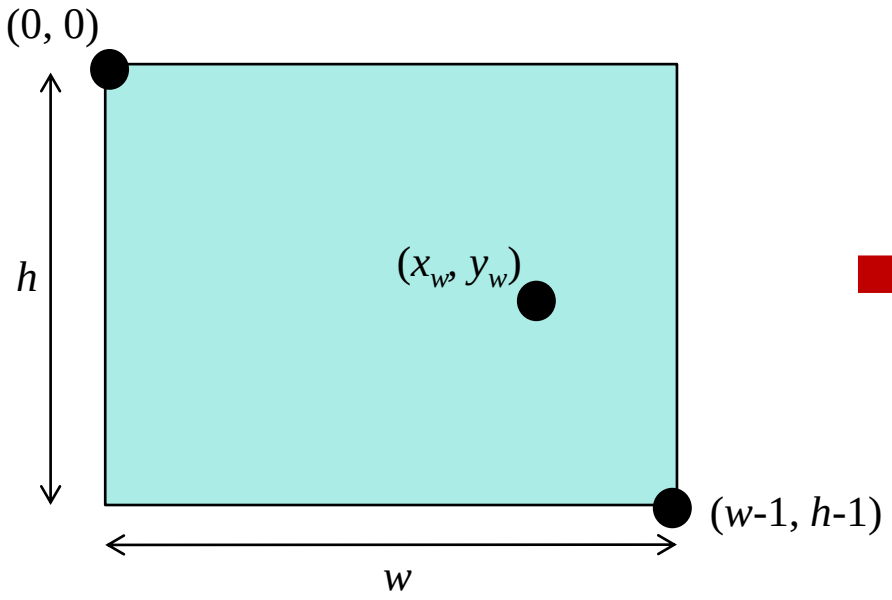
```
canvas.addEventListener("mousedown", function(e){
   if( e.button === 0 ) rotate = !rotate;
   if( e.button === 2 ) direction = -1*direction;
} );
```

Fyrirlestraæfingar

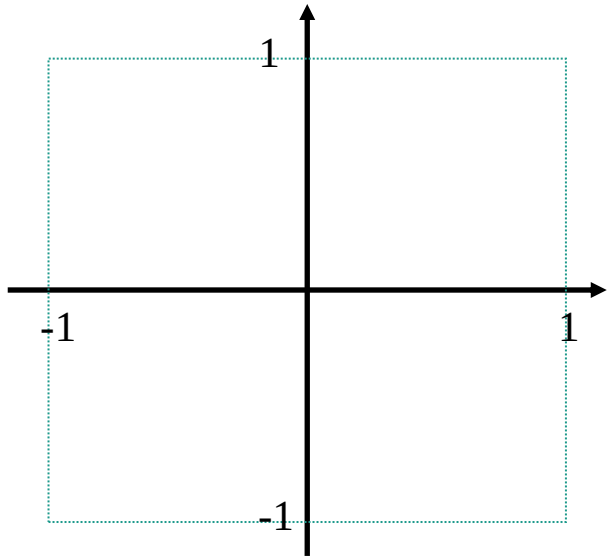
1. Í forritinu **`rotatingSquare1`** er breytan **`theta`** hækkuð í hverri ítrun. Hvaða áhætta er fólgin í því?
2. Sýnið atburðafall fyrir "**`mousedown`**", þannig að vinstri músarsmellur hækkar hraðabreytuna **`speed`** um 10%, en hægri smellur lækkar hana um 10%.
3. Ef striginn (*canvas*) er 100x100 og við smellum á skjápunkt (25,75), hvaða gildi hefur sá punktur í heimshnitum?

Staðsetning músar

- Þurfum að þýða skjáhnit yfir í heimshnit til að finna staðsetningu músar



Skjáhnit



Heimshnit

Skjáhnit hafa (0,0) í efra vinstra horn skjás

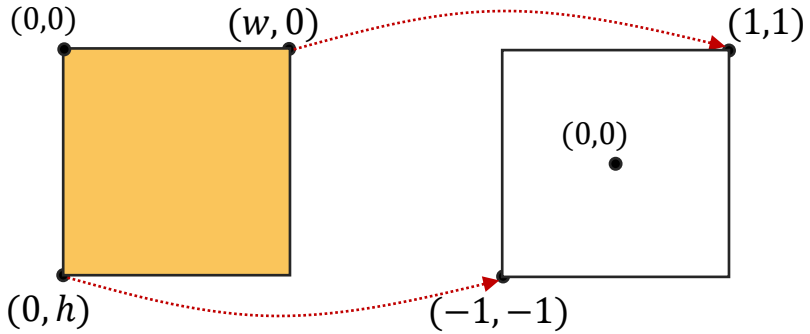
Formúlur fyrir breytingu á hnitum

- Hornpunktar:

$$(0,h) \rightarrow (-1,-1)$$

$$(w,0) \rightarrow (1,1)$$

Skjáhnit Heimshnit



- Almennt:

$$x = \frac{2 * x_w}{w} - 1 \qquad y = \frac{2 * (h - y_w)}{h} - 1$$

Sýniforritið `clickpoints`

- Hver músarsmellur bætir einum punktahnitum við í grafíkminnissvæðið

```
canvas.addEventListener("mousedown", function(e){
    gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer);

    // Calculate coordinates of new point
    var t = vec2(2*e.offsetX/canvas.width-1,
                 2*(canvas.height-e.offsetY)/canvas.height-1);

    // Add new point behind the others
    gl.bufferSubData(gl.ARRAY_BUFFER, 8*index, flatten(t));
    index++;
} );
```

Útreikningur á heimshnitum

Hver hnit eru 8 bæti (2 x `float`)

Sýniforritið spadi

- Nota músarhreyfingu til að stjórna spaða
  - En bara þegar músarhnappur er niðri

```
canvas.addEventListener("mousedown", function(e){
    movement = true;
    mouseX = e.offsetX;
} );
canvas.addEventListener("mouseup", function(e){
    movement = false;
} );
```

Leyfa spaðahreyfingu

Geyma upphafsstaðsetningu

```
canvas.addEventListener("mousemove", function(e){
    if(movement) {
        var xmove = 2*(e.offsetX - mouseX)/canvas.width;
        mouseX = e.offsetX;
        for(i=0; i<4; i++) {
            vertices[i][0] += xmove;
        }
    gl.bufferSubData(gl.ARRAY_BUFFER, 0, flatten(vertices));
} } );
```

Reikna hliðrun músar

Breytum hnitum í JS!

Sýniforritið `spadi-orvar`

- Nota örvalykla til að færa spaða
  - Þurfum að finna kóða fyrir örvalykla (keycode.info)

```
window.addEventListener("keydown", function(e){
    switch( e.keyCode ) {
        case 37: // vinstri ör
            xmove = -0.05; break;
        case 39: // hægri ör
            xmove = 0.05; break;
        default: xmove = 0.0;
    }
    for(i=0; i<4; i++) {
        vertices[i][0] += xmove;
    }
    gl.bufferSubData(gl.ARRAY_BUFFER, 0, flatten(vertices)); }
);
```

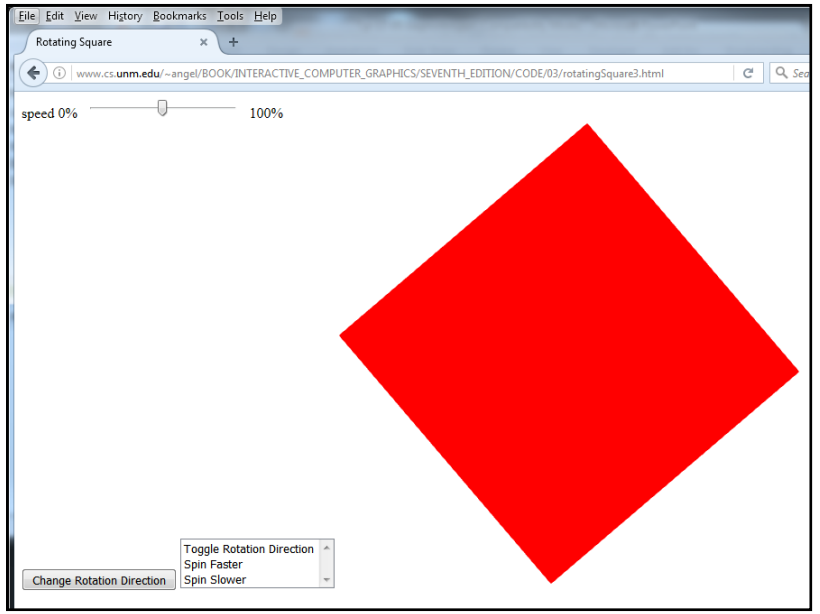
Kóði fyrir vinstri örvalykil er 37

Kóði fyrir hægri örvalykil er 39

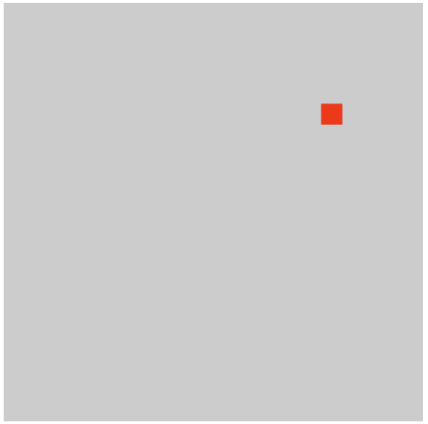
Hér breytum við hnitunum í JS!
Gætum líka sent **xmove** til GPU

Önnur sýnidæmi

- **rotatingSquare2**
  - Hnappur og valmynd
- **rotatingSquare3**
  - Hnappur, valmynd og sleði



- **box-bounce**
  - Lítill ferningur skoppar um strigann, hægt að breyta um hraða hans með upp/niður örvalyklum



Fyrirlestraæfingar

1. Í forritinu `rotatingSquare1` er breytan `theta` hækkuð í hverri ítrun. Hvaða áhætta er fólgin í því?
2. Sýnið atburðafall fyrir "`mousedown`", þannig að vinstri músarsmellur hækkar hraðabreytuna `speed` um 10%, en hægri smellur lækkar hana um 10%.
3. Ef striginn (*canvas*) er 100x100 og við smellum á skjápunkt (25,75), hvaða gildi hefur sá punktur í heimshnitum?