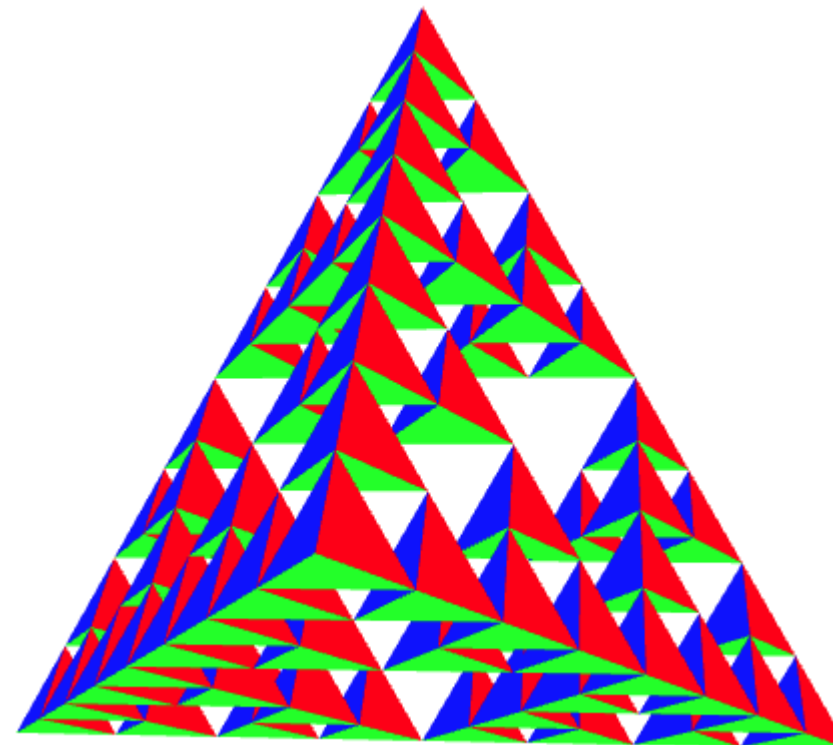


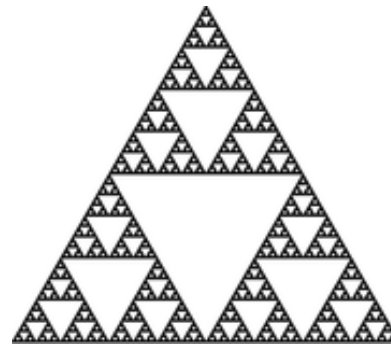
TÖL105M TÖLVUGRAFÍK

Fyrirlestur 3: Sierpinski

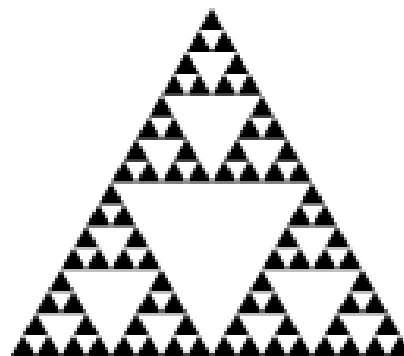
Hjálmtyr Hafsteinsson
Haust 2024



- Sierpinski þríhyrningurinn
 - Skilgreining
 - Aðferðir við að teikna
- Sierpinski í WebGL
 - "Okkar" aðferð
 - Mögulegar útfærslur
 - WebGL forrit



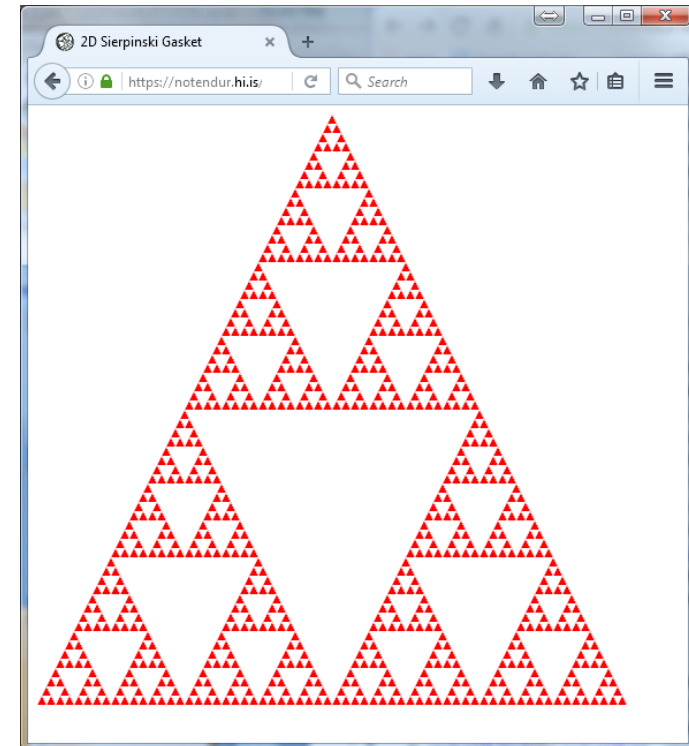
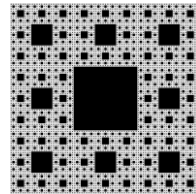
2.1



2.8

Sierpinski þríhyrningurinn

- Broti (*fractal*) sem er jafnhliða þríhyrningur
 - Samsettur úr öðrum jafnhliða þríhyrningum, "óendanlega" langt niður
 - Nefndur eftir pólska stærðfræðingnum [Waclaw Sierpinski](#) (1882-1969)
 - Mjög margar leiðir til að búa hann til (sjá [Stóru Sierpinski síðuna](#))
 - Til nokkrar aðrar útgáfur:
 - [Sierpinski teppið](#)
 - Sierpinski fjórflötungurinn



1. Hve margir fjórflötungar verða til í fyrstu ítrun á Sierpinski fjórflötungnum (sjá hreyfimynd efst á [síðu með sýnisforritum](#))?
2. Nefnið tvo ókosti við "*immediate mode*" aðferðina við að teikna flókna grafíkhloti
3. Ef við vildum færa Sierpinski þríhyrninginn til, eftir að punktarnir í honum eru komnir í grafíkminnið, hvar væri þá best að breyta punktahnitunum (í JS forritinu, hnútalitara eða bútalitara)?

Að smíða Sierpinski þríhyrning

- Byrja með jafnhliða þríhyrning, skipta honum í 4 þríhyrninga og taka burt þann sem er í miðjunni, endurtaka óendanlega oft:

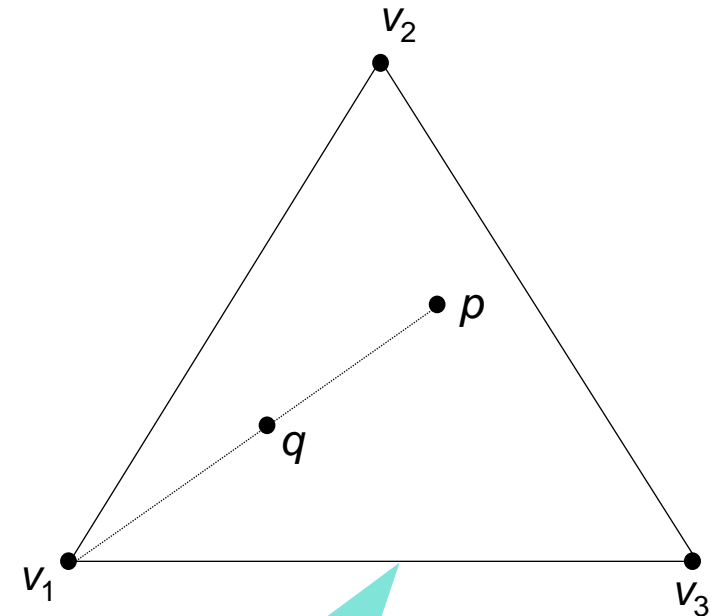


- Byrja með ferning, smækka hann og búa til 3 eintök af honum sem mynda turn, endurtaka óendanlega oft:



Okkar aðferð: Óreiðuleikurinn

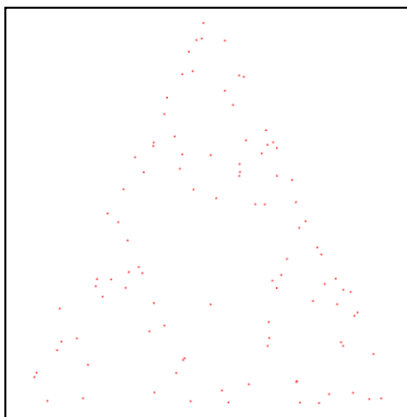
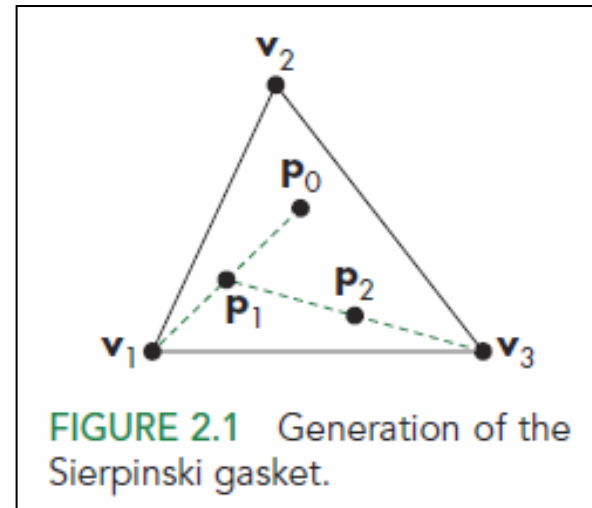
- Byrjum með jafnhliða þríhyrning úr punktunum v_1 , v_2 , v_3
 1. Velja upphafspunkt p inni í þríhyrningnum
 2. Velja af handahófi einn af hornpunktunum þremur
 3. Lita punkt q sem er mitt á milli p og valda hornpunktsins
 4. Setja punktinn q sem upphafspunktinn p og fara aftur í skref 2



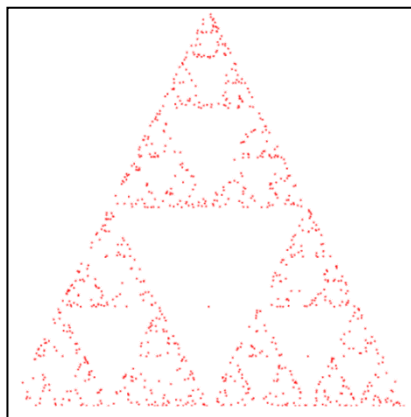
Ath: Teiknum bara punktana, engar línur!

Óreiðuleikurinn (*chaos game*)

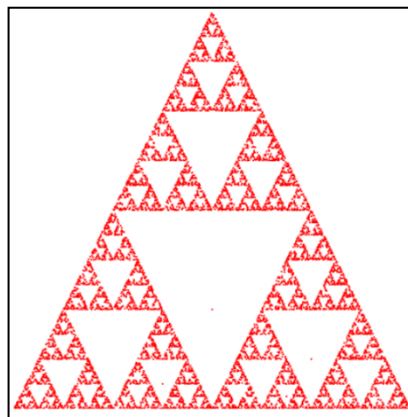
- Smátt og smátt verður til mikið af punktum og þeir mynda Sierpinski þríhyrninginn



100 punktar



1000 punktar



10000 punktar

Í WebGL forritinu birtast allir
punktarnir í einu!

- Þurfum að búa til mikinn fjölda punkta og teikna þá
- Skoðum nokkrar leiðir til að gera það:
 - Búa til punkt og teikna hann um leið (*immediate mode graphics*)
 - Búa til alla punktana og teikna þá alla í einu (*retained mode graphics*)
 - Þriðja leiðin
 - "Nýja" OpenGL - sú leið sem við notum

"Gamla" OpenGL

Leið I (*immediate mode*)

```
p = finna-fyrsta-punkt()  
FYRIR i=0 TIL 5000  
    q = næsti-punktur(p)  
    sýna-punktur(q)  
    p = q  
ENDIR
```

Sendum hvern punkt strax
(*immediately*) yfir til grafíkkorts

Svona er grafíkin í "gamla"
OpenGL (útgáfa < 3.0)

Vandamál:

Dýrt að senda lítið gagnamagn til grafíkkorts
(gagnasending verður flöskuháls)

Leið II (*retained mode*)

```
p = finna-fyrsta-punkt()  
FYRIR i=0 TIL 5000  
    q = næsti-punktur(p)  
    geyma-punkt(q)  
    p = q  
ENDIR  
sýna-alla-punkta()
```

Geymum (*retain*)
punkta í aðalminni

Sendum alla punkta í einu og
þeir eru allir birtir samtímis

Vandamál:

Ef við viljum sýna sömu punktana
aftur þá þarf að senda þá aftur

```
p = finna-fyrsta-punkt()  
FYRIR i=0 TIL 5000  
    q = næsti-punktur(p)  
    geyma-punkt(q)  
    p = q  
ENDIR  
senda-punkta-til-GPU()  
sýna-punkta-á-GPU()
```

Geymum (*retain*)
punkta í aðalminni

Setjum alla punkta á
tiltekinn stað í grafíkminni

Lesa punktana úr
grafíkminni og birta þá

Þetta er aðferðin sem verður
notuð í öllum okkar forritum

Kostir:

- Hægt að endursýna punktana án þess að senda þá aftur
- Getum látið grafíkkort breyta punktunum (litarar)

1. Hve margir fjórflötungar verða til í fyrstu ítrun á Sierpinski fjórflötungnum (sjá hreyfimynd efst á síðu með sýnisforritum)?
2. Nefnið tvo ókosti við "*immediate mode*" aðferðina við að teikna flókna grafíkhloti
3. Ef við vildum færa Sierpinski þríhyrninginn til, eftir að punktarnir í honum eru komnir í grafíkminnið, hvar væri þá best að breyta punktahnitunum (í JS forritinu, hnútalitara eða bútalitara)?

- Gróft skipulag JS kóða:
 - Upphafsstilla strigann (*canvas*)
 - Búa til alla punktana í fylki (hér 5000 punktar)
 - Upphafsstilla WebGL:
 - Hlaða inn liturum og senda til GPU
 - Skilgreina minni á GPU
 - Senda punkta yfir
 - Tengja punkta við litarabreytur
 - Kalla á teiknifall (**render()**)

Helsta flækjan í OpenGL

Upphafsstilling á striga

Fylkið sem geymir punktana áður en þeir sendir yfir til grafíkkorts

```
var gl;  
var points;  
  
var NumPoints = 5000;  
  
window.onload = function init()  
{  
    var canvas = document.getElementById( "gl-canvas" );  
  
    gl = WebGLUtils.setupWebGL( canvas );  
    if ( !gl ) { alert( "WebGL isn't available" ); }  
  
    ...  
}
```

Fallið `init()` er keyrt þegar vafrinn hefur hlaðinn vefsíðunni

Upphafsstilla teiknisvæðið (strigann) og gefa villu ef það tókst ekki

Búa til punkta í fylki

Hornpunktarnir þrír.
Hnitakerfið er frá -1 til 1 á x- og y-ás

```
...  
var vertices = [vec2( -1, -1 ), vec2( 0, 1 ), vec2( 1, -1 )];  
  
// Specify a starting point p for our iterations  
var u = add( vertices[0], vertices[1] );  
var v = add( vertices[0], vertices[2] );  
var p = scale( 0.25, add( u, v ) );  
points = [ p ];  
  
// Compute new points  
for ( var i = 0; points.length < NumPoints; ++i ) {  
    var j = Math.floor(Math.random() * 3);  
    p = add( points[i], vertices[j] );  
    p = scale( 0.5, p );  
    points.push( p );  
}  
...
```

Finnum upphafspunkt sem er inni í
þríhyrningnum, hér (-0.25, -0.5)

Hann fer í fylkið `points`

Velja hornpunkt af handahófi

Finna punkt mitt á milli hans og
síðasta punkts

Setja nýja punktinn aftast í fylkið

Upphafstillla WebGL og hlaða litara

```
...  
    // Configure WebGL  
  
    gl.viewport( 0, 0, canvas.width, canvas.height );  
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );  
  
    // Load shaders and initialize attribute buffers  
  
    var program = initShaders( gl, "vertex-shader", "fragment-shader" );  
    gl.useProgram( program );  
...
```

Skilgreina víddir sjónlugga

Hreinsilitur (þ.e.
bakgrunnur) er hvítur

Fallið `initShaders()` kemur frá
höfundum bókarinnar. Það hleður
inn hnútalitara og bútalitara

Virkja litarana

Senda punkta yfir og tengja þá við litara

...

// Load the data into the GPU

var bufferId = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, bufferId);

gl.bufferData(gl.ARRAY_BUFFER, flatten(points), 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);

...

Skilgreina minnissvæði á grafikkorti

Taka fram gerð minnisins (venjul. fylki)

Færa gögnin úr `points` fylkinu yfir í minnissvæðið

Fallið `flatten` breytir úr `vec2` yfir í einfalt fylki

Tengja minnissvæðið við breytuna `vPosition` í hnútalitaranum (Skoðum þetta nánar síðar)

```
...  
    render();  
};  
  
function render() {  
    gl.clear( gl.COLOR_BUFFER_BIT );  
    gl.drawArrays( gl.POINTS, 0, points.length );  
}
```

Síðasta skipun í `init()` fallinu er að kalla á teiknifallið `render()`

Mjög einföld útgáfa af teiknifalli:
- Hreinsa teiknisvæði
- Teikna alla punktana

Fyrstu forritin okkar munu aðeins teikna grafíkina einu sinni, en þegar við förum í hreyfimyndir, þá er teiknifallið framkvæmt mörgum sinnum á sek.

Í hreyfimyndunum er líka gott að hafa alla punktana þegar á grafíkkortinu, því þá þarf kannski aðeins að breyta einum stika og teikna svo aftur.

Teikniskipunin `gl.drawArrays`

- Teiknar punkta í núverandi minnissvæði (á GPU) samkvæmt því sem fyrsta viðfangið skilgreinir

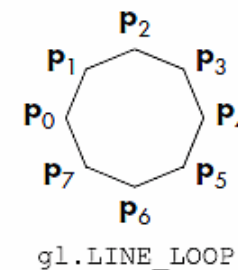
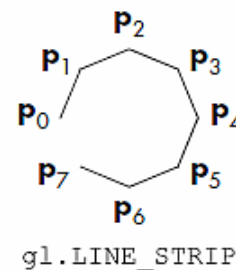
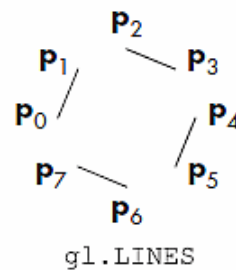
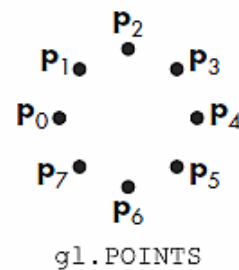
```
gl.drawArrays( gl.POINTS, 0, points.length );
```

Hvað á að teikna

Fyrsti punktur í minnissvæði

Hvað á að teikna marga punkta

- Aðrir möguleikar á teikningu:



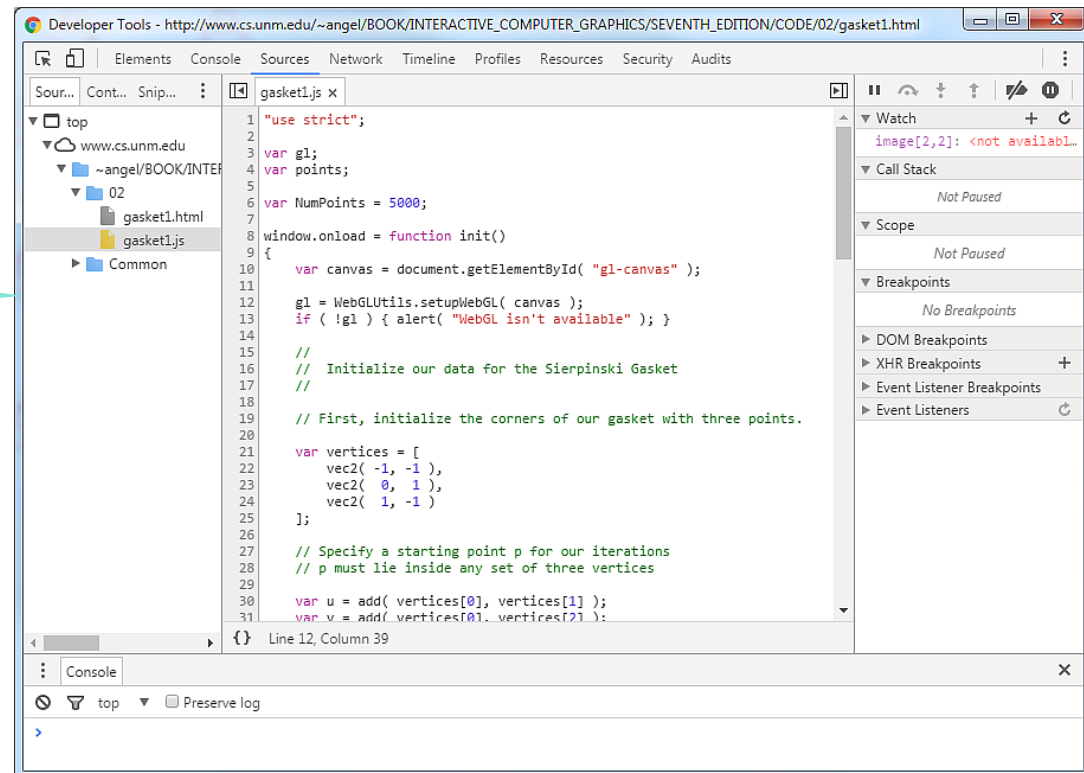
- Á heimasíðu bókarinnar:
 - [gasket1.html](#) - notar skrárnar sem eru í [Common](#) möppunni
- Til að keyra á eigin tölvu:
 - Ná í skrárnar í Common möppunni:
 - [MV.js](#) - Javascript forritasafn fyrir fylkja- og vigurvinnslu
 - [webgl-utils.js](#) - föll frá Google til að upphafsstilli WebGL
 - [initShaders.js](#) - fall frá bókarhöfundum til að hlaða inn liturum
 - Setja möppur upp á svipaðan hátt og gert á heimasíðu bókarinnar
 - Annars þarf að breyta slóðum í innskotssetningum í HTML-skrám

Aflúsun WebGL forrita

- Google Chrome:
 - Nota [Chrome DevTools](#)
 - Smella á "Sources" flipann og finna JS skránnu

Hægt að keyra línu fyrir línu og skoða innihald breyta

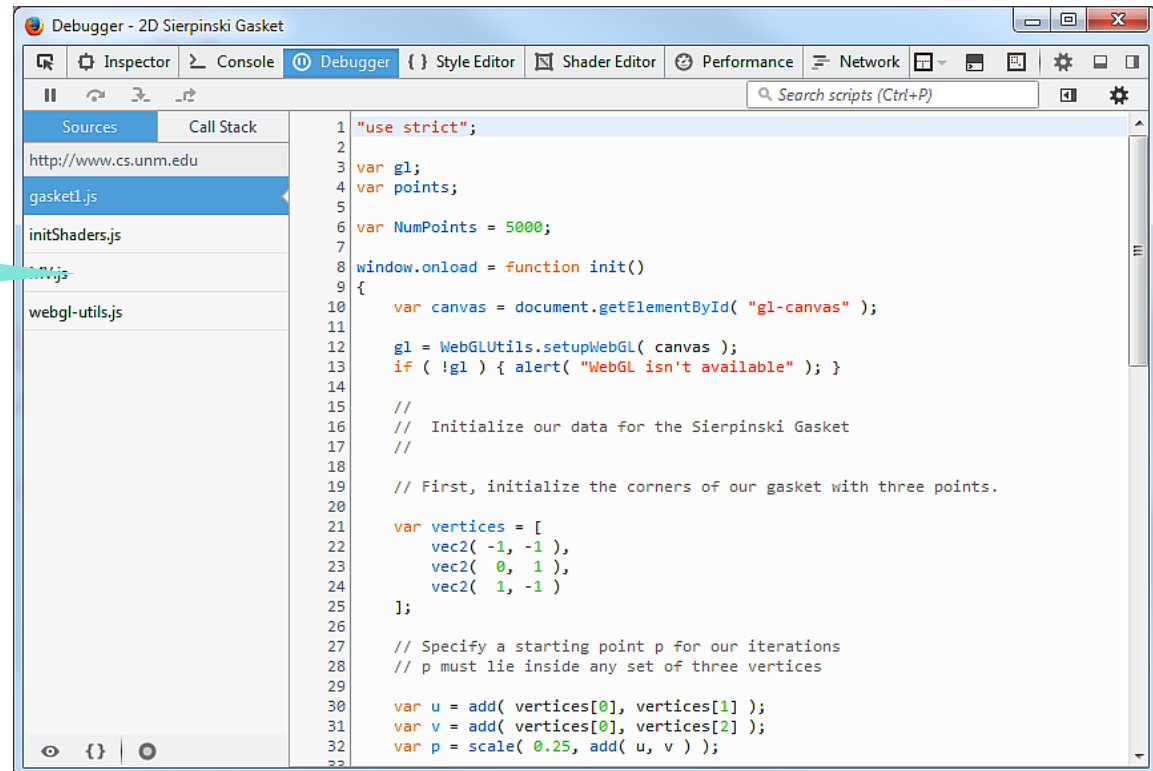
Sjáum ekki inní grafíkminnið og ekki auðvelt að rekja keyrslu á liturum



- Firefox:
 - Nota Page Inspector
 - Smella á "*Debugger*" flipann

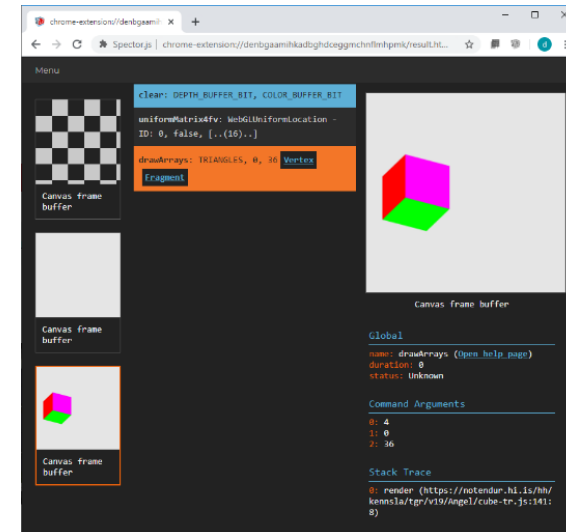
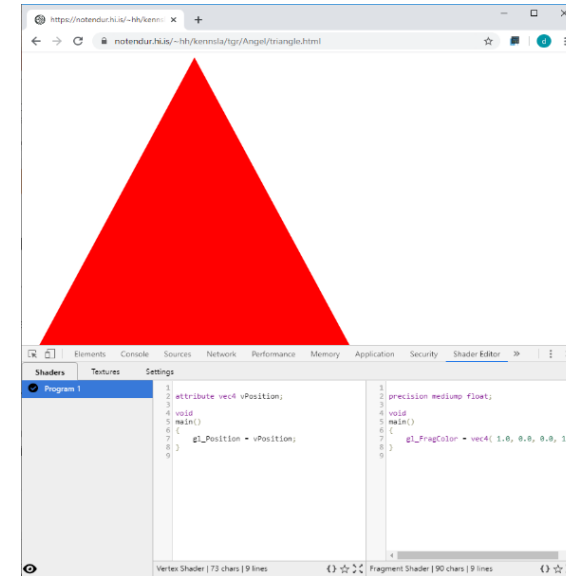
Hægt að keyra línu fyrir línu og skoða innihald breyta


Sjáum ekki inní grafíkminnið en hægt að setja upp "*Shader Editor*" til að vinna með litara



```
1 "use strict";
2
3 var gl;
4 var points;
5
6 var NumPoints = 5000;
7
8 window.onload = function init()
9 {
10     var canvas = document.getElementById( "gl-canvas" );
11
12     gl = WebGLUtils.setupWebGL( canvas );
13     if ( !gl ) { alert( "WebGL isn't available" ); }
14
15     //
16     // Initialize our data for the Sierpinski Gasket
17     //
18
19     // First, initialize the corners of our gasket with three points.
20
21     var vertices = [
22         vec2( -1, -1 ),
23         vec2( 0, 1 ),
24         vec2( 1, -1 )
25     ];
26
27     // Specify a starting point p for our iterations
28     // p must lie inside any set of three vertices
29
30     var u = add( vertices[0], vertices[1] );
31     var v = add( vertices[0], vertices[2] );
32     var p = scale( 0.25, add( u, v ) );
```

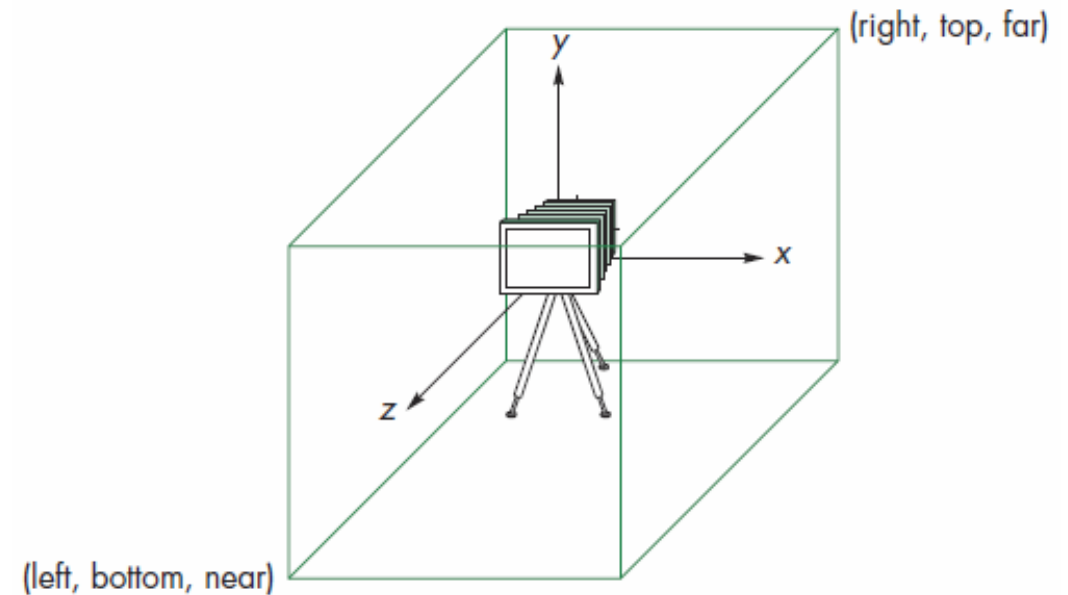
- Eingöngu til sem viðbætur við vafra
 - [Shader Editor](#) fyrir Google Chrome
 - Virkar best í hreyfiforritum, því þar er hægt að breyta liturum og sjá strax áhrif
 - [spector.JS](#) fyrir Chrome og Firefox
 - Öflugt tól til að skoða hvað er að gerast inni í liturunum
 - Frá [Babylon.js](#), sem er 3D grafíkvél í WebGL



- Vinnum með heimshnit (*world coordinates*) í JS forritinu
 - Skilgreinum stærð og staðsetningu hluta í þeim
 - **Sjónrúm** (*viewing volume*) skilgreint í heimshnitum  Líka *object coordinates*
- Að lokum eru notuð skjáhnit (*window coordinates*) til að birta mynd á skjá
 - Á milli þessara hnitakerfa eru önnur hnitakerfi sem WebGL notar í útreikningum

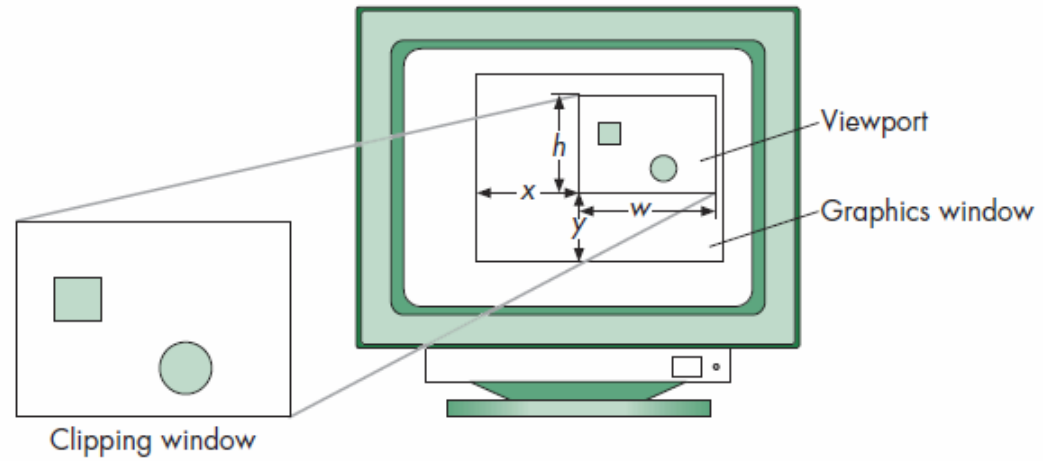
Sjónrúm (*viewing volume*)

- Skilgreinir hvað hægt er að sjá
 - Allt annað er klippt í burtu
 - Sjálfgefið sjónrúm er teningur með miðju í $(0, 0, 0)$ og hliðar af lengd 2.0
 - Myndavélin í WebGL er í $(0, 0, 0)$ og horfir niður eftir $-z$ -ásnum
 - Í upphafi erum við í tvívídd og notum því **sjónferning** með $z = 0$



Sjóngluggi (*viewport*)

- Þegar við teiknum á grafíkgluggann (þ.e. strigann) þurfum við ekki að nota hann allan
 - Skilgreinum sjónglugga með **viewport**-fallinu
- ```
gl.viewport(x, y, w, h);
```
- Getum notað það til að skilgreina teiknisvæði í grafíkglugganum fyrir mismunandi sýn á heiminn



1. Hve margir fjórflötungar verða til í fyrstu ítrun á Sierpinski fjórflötungnum (sjá hreyfimynd efst á síðu með sýnisforritum)?
2. Nefnið tvo ókosti við "*immediate mode*" aðferðina við að teikna flókna grafíkhloti
3. Ef við vildum færa Sierpinski þríhyrninginn til, eftir að punktarnir í honum eru komnir í grafíkminnið, hvar væri þá best að breyta punktahnitunum (í JS forritinu, hnútalitara eða bútalitara)?