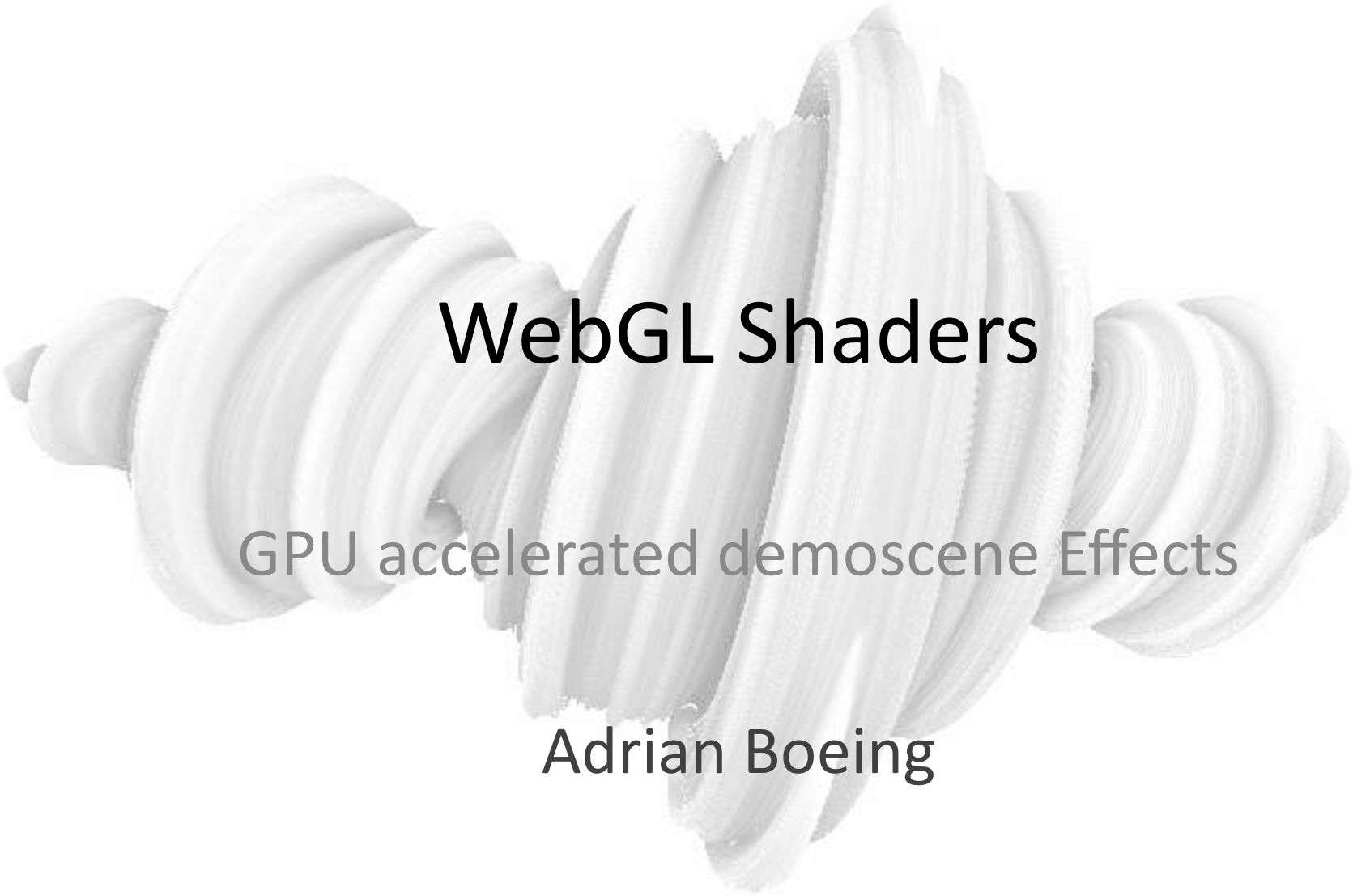


WebGL Shaders

GPU accelerated demoscene Effects

Adrian Boeing

Introduction

- About Me
- The Demoscene
- WebGL & Shaders
- Circles & Trig,

& some demo effects

*Note: Download **Chrome** or Firefox or **Webkit**!*

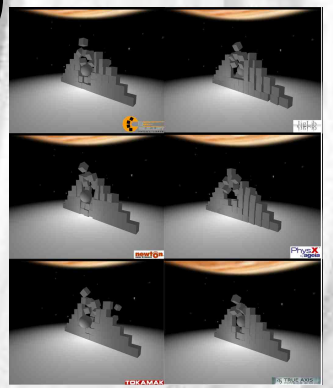
About Me

- Undergrad UWA (Engineering)

PIXEL JUICE

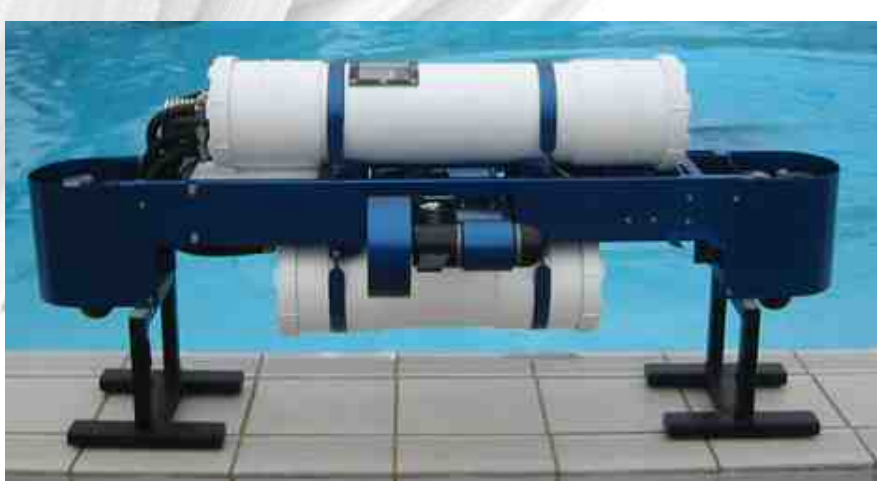
About Me

- Postgrad UWA (Simulation & Robotics)



THE UNIVERSITY OF WESTERN AUSTRALIA

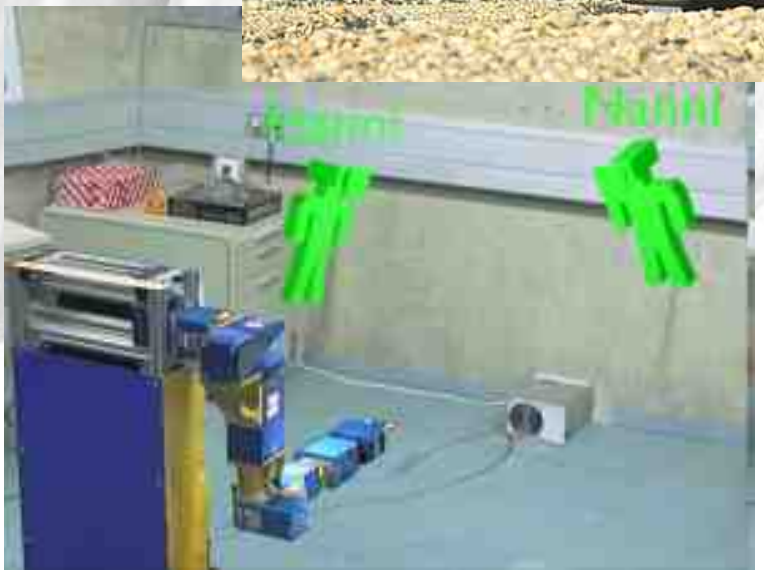
Raytheon



About Me

- TUM/UniBW

der Bundeswehr Universität München

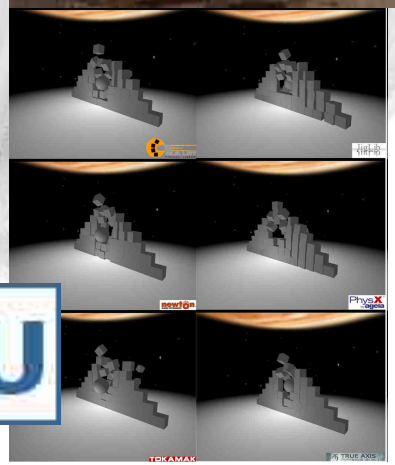


About Me

- ECU/WAMbot

wambot
Western Australia MAGIC 2010 Robotics Team
http://www.wambot.org/

iVEC

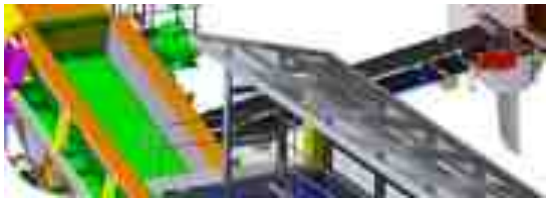


AUSTRALIA ECU EDITH COWAN UNIVERSITY

GPGPU

About Me

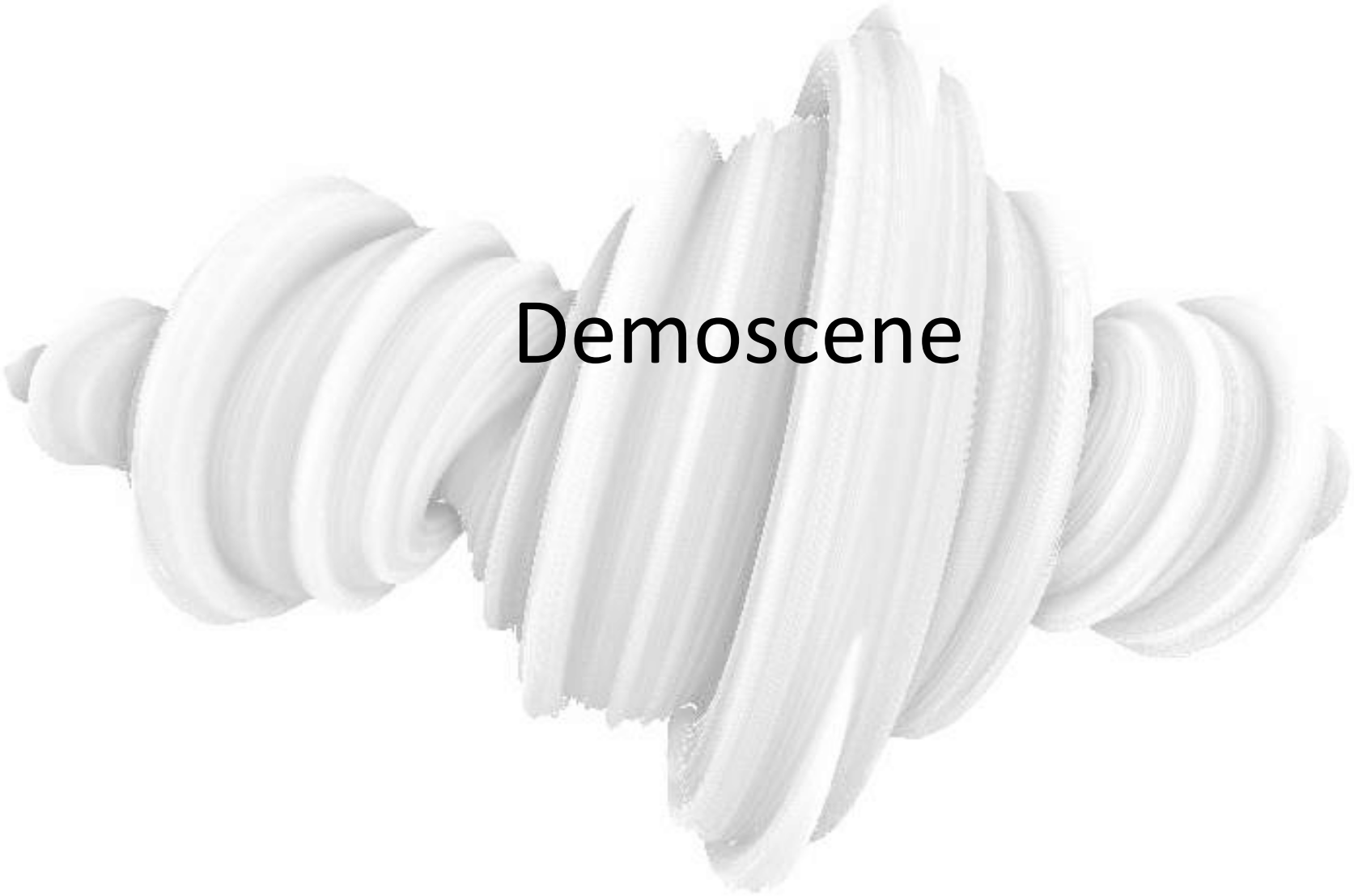
- Transmin/Rocklogic



TRANSMIN
Australian Engineering Worldwide

ROCKLOGIC™

http://www.youtube.com/watch?v=v1AJ_OBJUpY

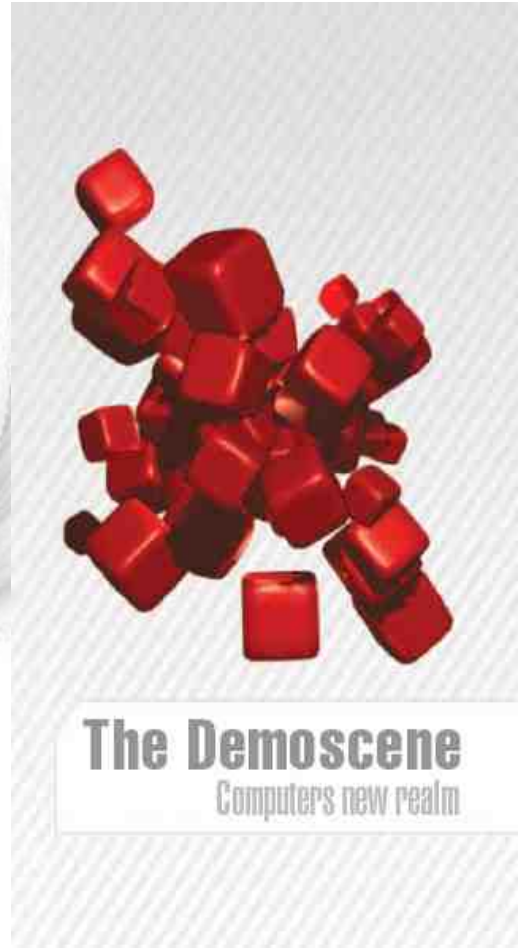


Demoscene

Demoscene

- Demos are a fusion of:
  - Code
  - Art
  - Music
  - Design

… to generate a jaw-dropping realtime audiovisual production



Demoscene (nVidia 2008)

http://www.youtube.com/watch?v=PidTKpKLYZM

Demoscene Culture

- Demo parties
- European centric
- Graphics & Games & Film Companies

  nVidia, Pixar, Dice

scene.org

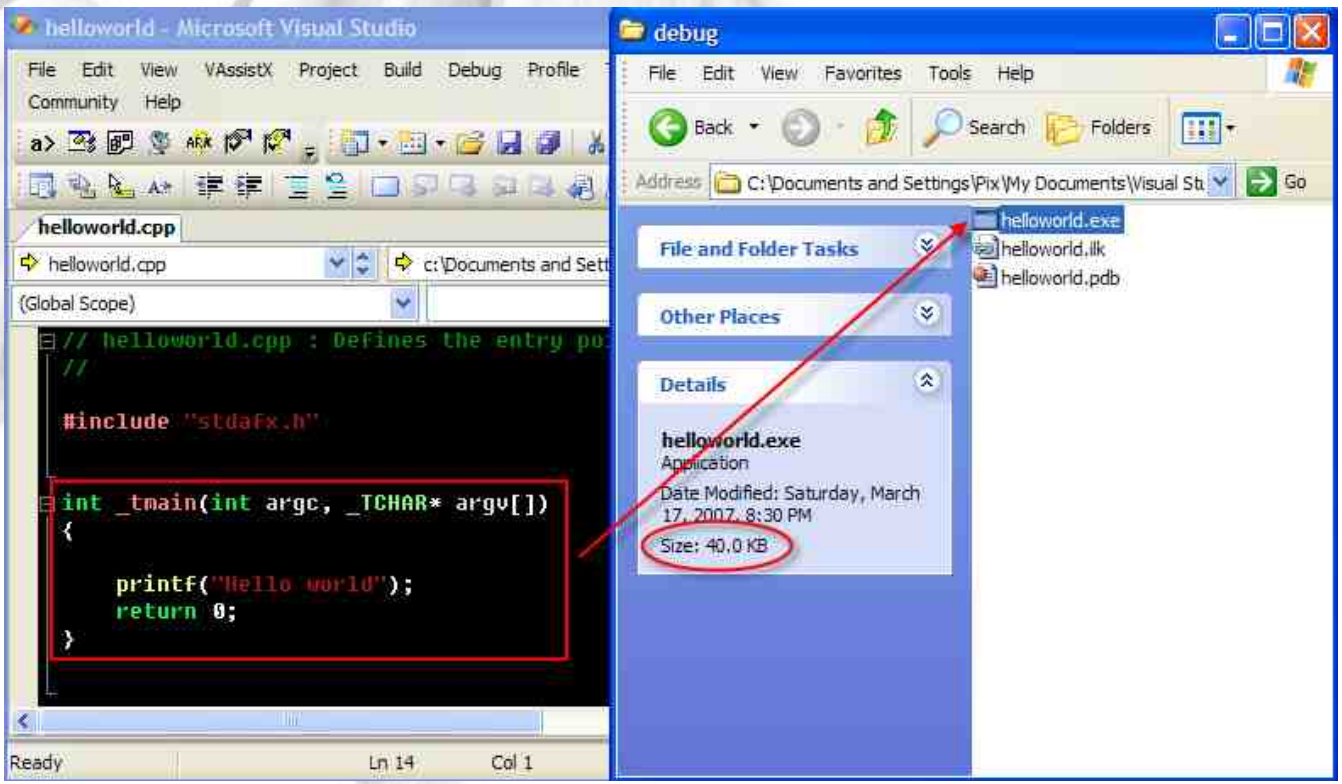
pouet.net

Demoscene

- Three key varieties:
- Demo
- Intro (64k)
- Intro (4k)

Small?

Algorithms!



Demoscene: History

- Originated from ‘crack screens’
- C64, Amiga, traditional platforms
- IBM PC Scene saw serious growth in mid-90’s
  - Future Crew, 1994 – Second Reality – 486, 33mhz
  - Farbraush, 2000 – the product – P3 – 500mhz

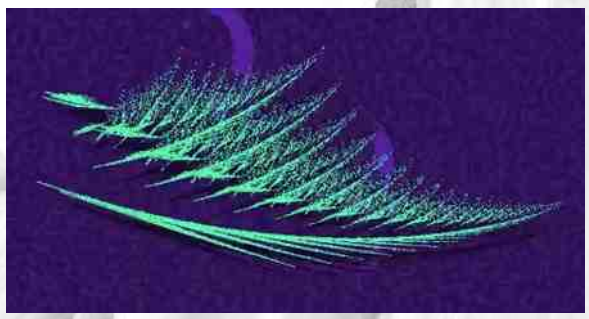
Less CPU/GPU than the original iPhone!

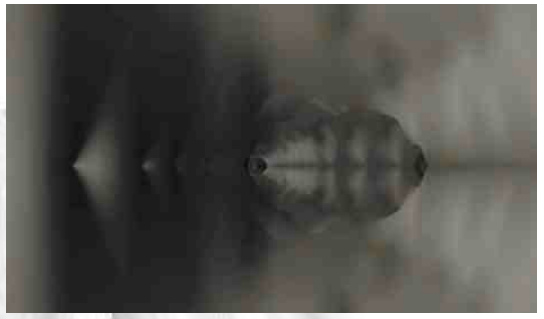
Demoscene: History [Verses/EMF]

http://www.youtube.com/watch?v=eYWCSHqOikY

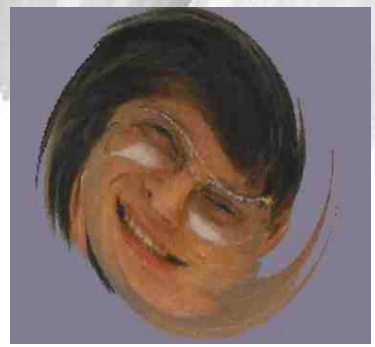
The 90's were dominated by 2d effects. Later by 3d graphics and raytracing, now GPU shaders, image processing and radiosity

Demoscene Effects (1994)

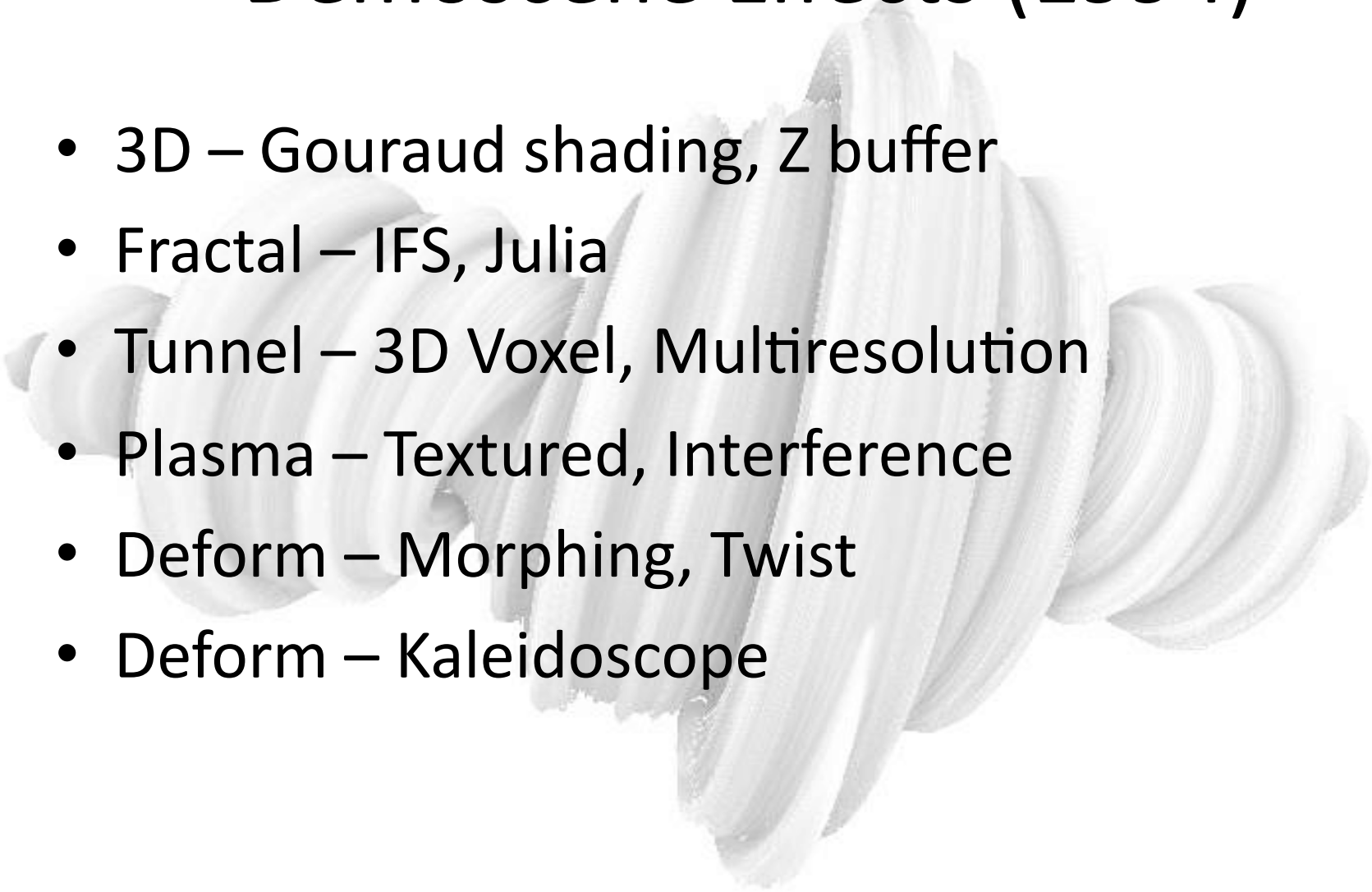




Verses by *EMF*



Demoscene Effects (1994)



- 3D – Gouraud shading, Z buffer
- Fractal – IFS, Julia
- Tunnel – 3D Voxel, Multiresolution
- Plasma – Textured, Interference
- Deform – Morphing, Twist
- Deform – Kaleidoscope


WebGL

WebGL

- Uses Javascript to generate hardware accelerated graphics within a web browser (ie OpenGL in a browser!)
  - Based on OpenGL ES
- Version 1.0 released March, 2011
  - Chrome & Firefox
- Safari & Opera & Android & iPhone WIP

http://doesmybrowsersupportwebgl.com/

WebGL

- WebGL makes heavy use of shaders
  - Avoid pure javascript, as it is much slower than doing calculations in WebGL shaders
- WebGL Pipeline:

Vertex Operation

⇓

Rasterization

⇓

Fragment Operation

⇓

FrameBuffer

WebGL: A triangle

```
<html><head>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
<script type="text/javascript">
  var fragmentShaderString =
  '#ifdef GL_ES                                  \n\
    precision highp float;                       \n\
    #endif                                       \n\
    void main(void) {                            \n\
      gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0); \n\
    }                                            \n\';
  var vertexShaderString =
  'attribute vec3 aVertexPosition;               \n\
    void main(void) {                            \n\
      gl_Position = vec4(aVertexPosition, 1.0);\n\
    }                                            \n\';
  var gl;  var shaderProgram;
  function initGL(canvas) {
    try {
      gl = canvas.getContext("experimental-webgl");
    } catch(e) {}
  }
  function getShader(gl, str, shaderType) {
    var shader = gl.createShader(shaderType);
    gl.shaderSource(shader, str);
    gl.compileShader(shader);
    return shader;
  }
  function initShaders() {
    var fragmentShader = getShader(gl,
      fragmentShaderString,gl.FRAGMENT_SHADER);
    var vertexShader = getShader(gl,
      vertexShaderString,gl.VERTEX_SHADER);
    shaderProgram = gl.createProgram();
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);
    gl.useProgram(shaderProgram);
    shaderProgram.vertexPositionAttribute =gl.getAttribLocation
      (shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
  }
```

```
  var triangleVertexPositionBuffer;
  function initBuffers() {
    triangleVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    var vertices = [
         0.0,  1.0,  0.0,
        -1.0,  -0.5,  0.0,
         1.0,  -0.5,  0.0
    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array
      (vertices),gl.STATIC_DRAW);
  }
  function drawScene() {
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
      3,gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLES, 0, 3);
  }
  function webGLStart() {
    var canvas = document.getElementById("lesson01-canvas");
    initGL(canvas);
    initShaders();
    initBuffers();
    drawScene();
  }
</script>
</head>
<body onload="webGLStart();">
  <canvas id="lesson01-canvas" style="border: none;"
      width="500"height="500"></canvas>
</body>
</html>
```

http://www.LearningWebGL.com/

WebGL Shaders

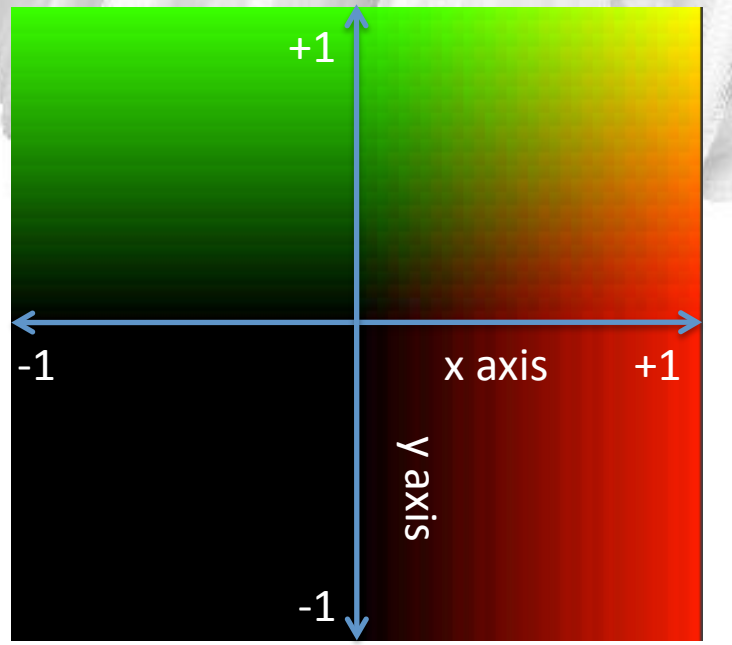
```
var fragmentShaderString =
'#ifdef GL_ES
 \n\
  precision highp float;
 \n\
  #endif
 \n\
  void main(void) {
 \n\
    gl_FragColor = vec4(1.0,
 1.0, 1.0, 1.0); \n\
  }
```

```
var vertexShaderString =
'attribute vec3
 aVertexPosition;
 \n\
  void main(void) {
 \n\
    gl_Position = vec4
 (aVertexPosition, 1.0);\n\
  }
```

Very similar to C/C++, Java, Cg, DirectX shaders, etc.

Let’s Start

- The sample code draws a quad, aligned with unit-screen space.
- Let’s assign some colors to the x & y coordinates. Center 0,0

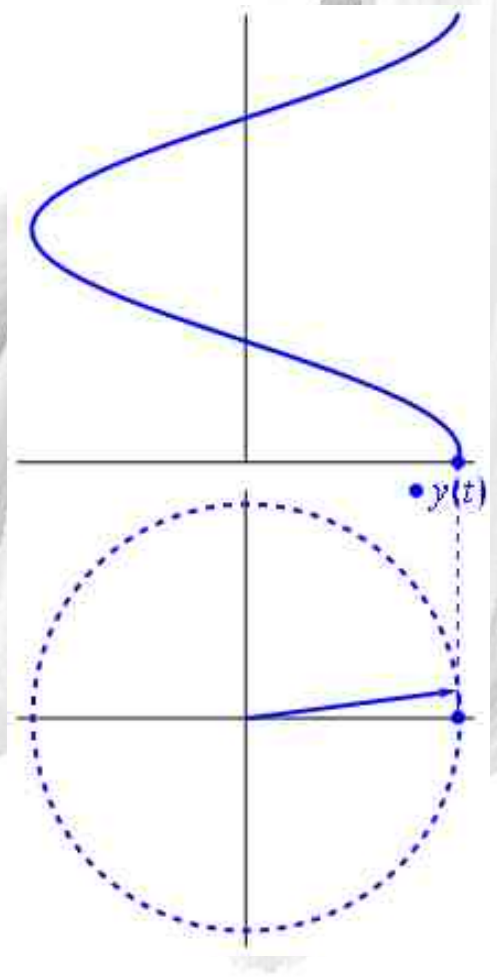


In WebGL Shader code

```
void main(void)
{
vec2 p = -1.0 + 2.0 * gl_FragCoord.xy /
   resolution.xy;
gl_FragColor = vec4(p.x,p.y,0.0,1.0);
}
```

WebGL Demo Effect: Plasmas

Sin & Cos



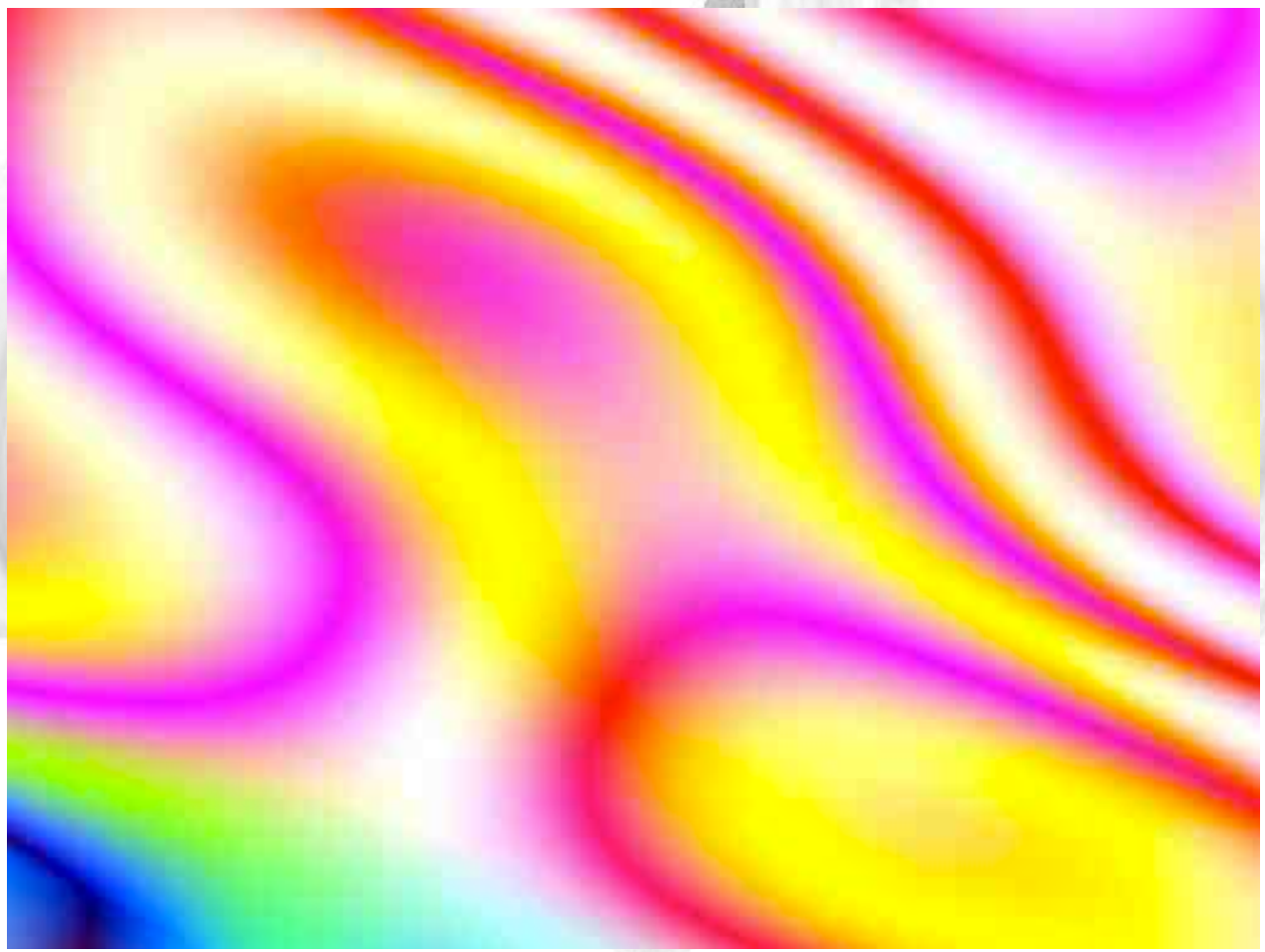
Try writing a shader where the red value is a sin-function of the x coordinate.
e.g.

red = sin(x)

Try playing with the frequency (w), amplitude (A) and phase (p).

$$y(t) = A \cdot \sin(\omega t + \varphi)$$

Plasma effect



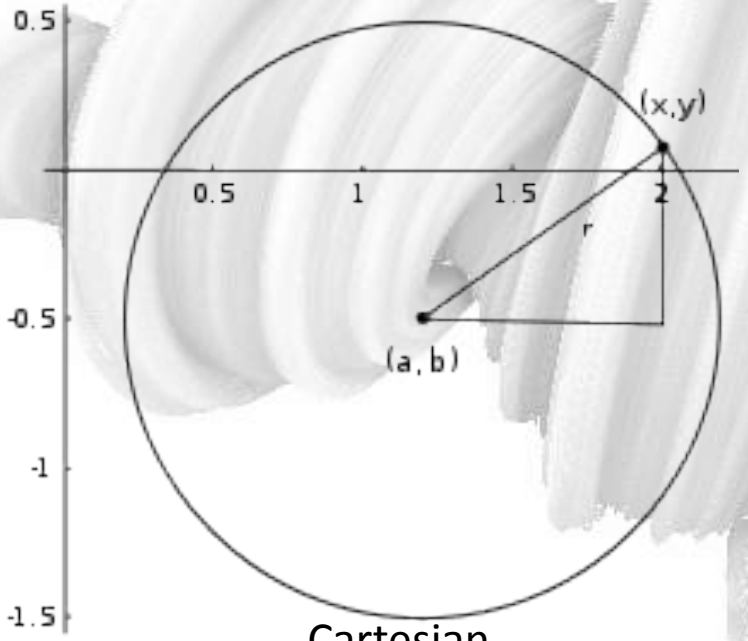
(Viktor Korsun)

Simple WebGL Plasma

```
precision highp float;
uniform vec2 resolution;
uniform float time;

void main(void) {
vec2 p = -1.0 + 2.0 * gl_FragCoord.xy / resolution.xy;
float cossin1 = cos(p.x*5.0+time)+sin(p.y*7.0-time)+sin(time);
float cossin2 = cos(p.y*3.0+time)+sin(p.x*9.0-time)-cos(time);
float cossin3 = cos(p.x*11.0+time)+0.5*sin(p.y*3.0+time)+cos(time);
gl_FragColor = vec4(cossin1*sin(p.x),cossin2*sin(p.y),cossin3*sin(p.x),
    1.0);
}
```

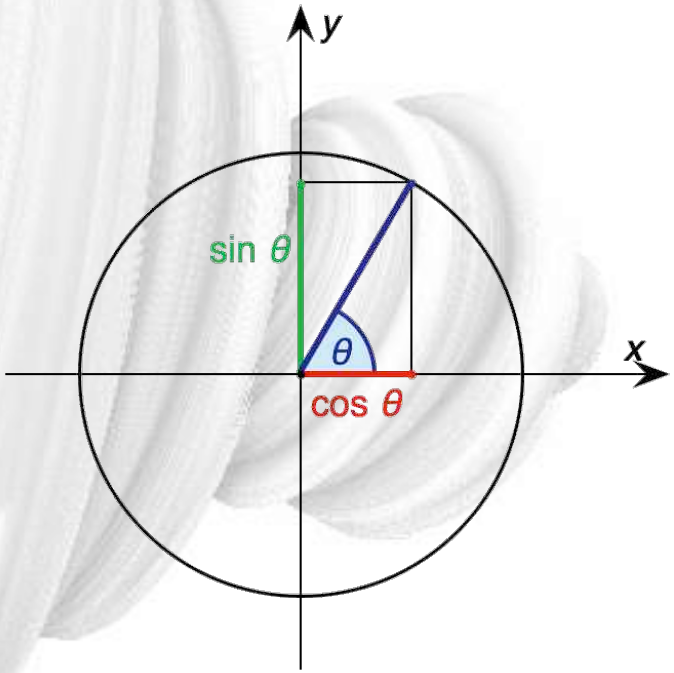
Sin & Cos & Circles

- Circle can also be expressed with Cartesian Coordinates



Cartesian

$$(x-a)^2 + (y-b)^2 = r^2.$$



Parametric

$$x = a + r\,\cos t,$$
$$y = b + r\,\sin t$$

Can you generate Circles?

Single, continuously increasing radius

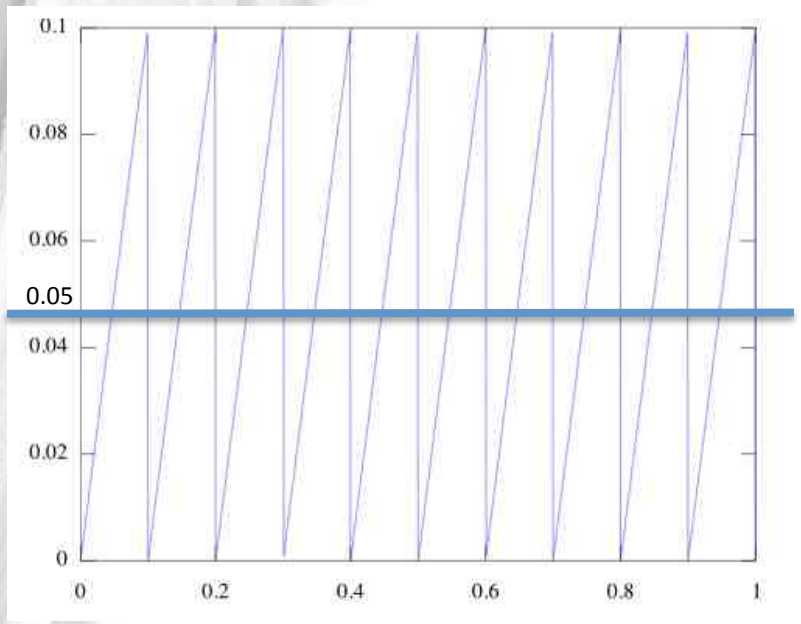
Multiple rings (hint: bands of circles)

http://adrianboeing.blogspot.com/2011/01/xor-demoeffect-in-webgl.html

Simple WebGL Circles

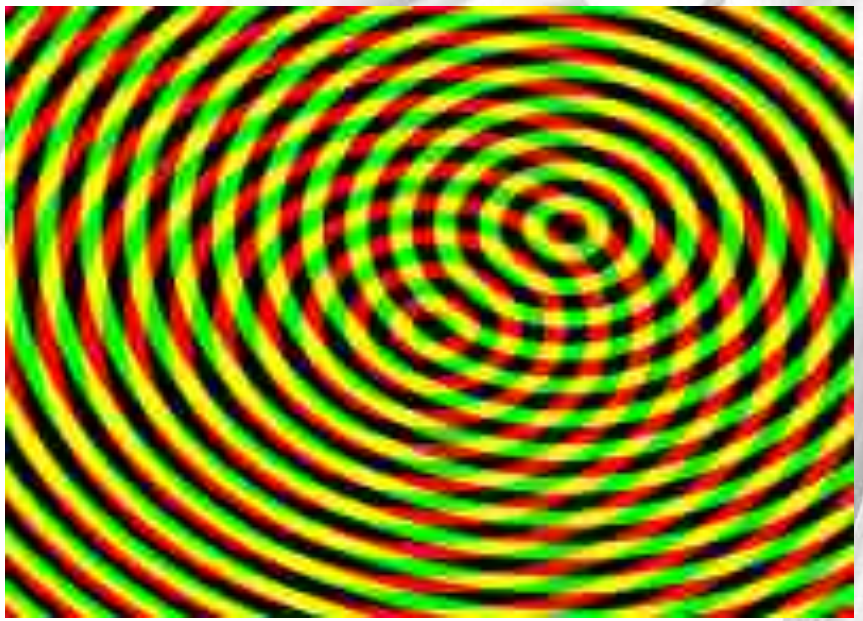
```
void main(void)
{
vec2 p = -1.0 + 2.0 *
    gl_FragCoord.xy /
    resolution.xy;
float radius = sqrt(p.x*p.x
    + p.y*p.y);
gl_FragColor = vec4
    (radius,0.0,0.0,1.0);
}
```

- bool toggle = mod (radius,0.1)>0.05;



http://adrianboeing.blogspot.com/2011/01/xor-demoeffect-in-webgl.html

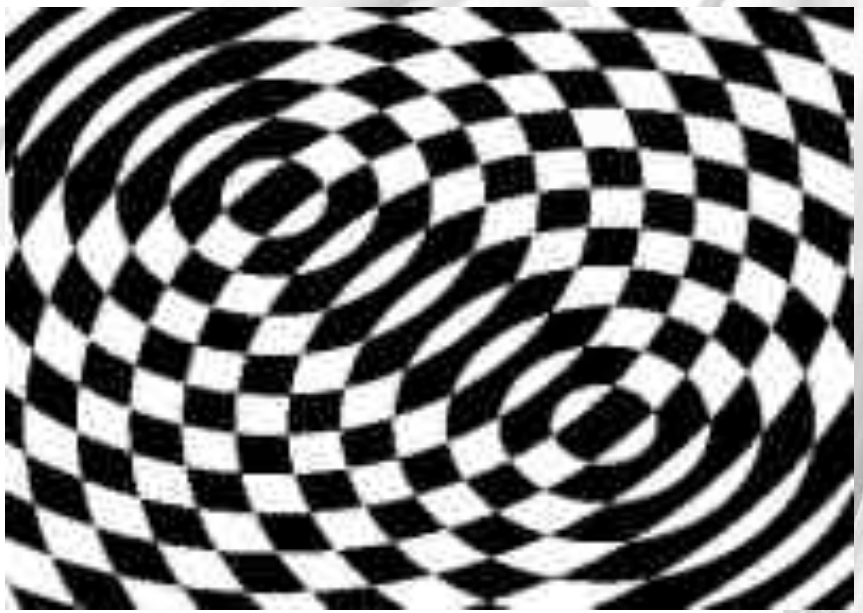
Two WebGL Circles



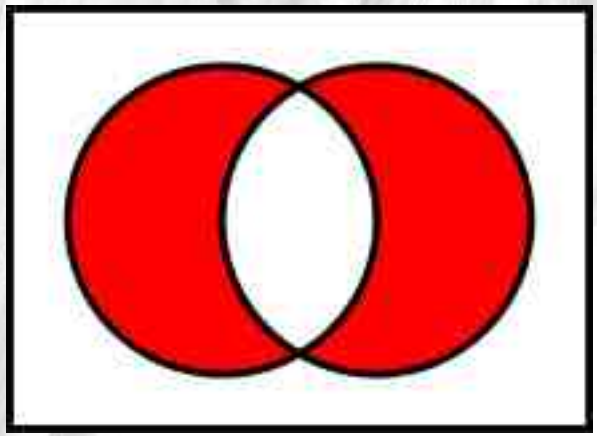
```
void main(void) {
vec2 p = -1.0 + 2.0 * gl_FragCoord.xy /
      resolution.xy;

vec2 offset2 = vec2(0.3,0.3);

float radius1 = sqrt(dot(p,p));
float radius2 = sqrt(dot(p-offset2,p-
      offset2));

bool toggle1 = mod(radius1,0.1)>0.05;
bool toggle2 = mod(radius2,0.1)>0.05;

gl_FragColor = vec4
      (toggle1,toggle2,0.0,1.0);
}
```

http://adrianboeing.blogspot.com/2011/01/xor-demoeffect-in-webgl.html

XOR bit plasma



A	B	A xor B
T	F	T
F	T	T
T	T	F
F	F	F



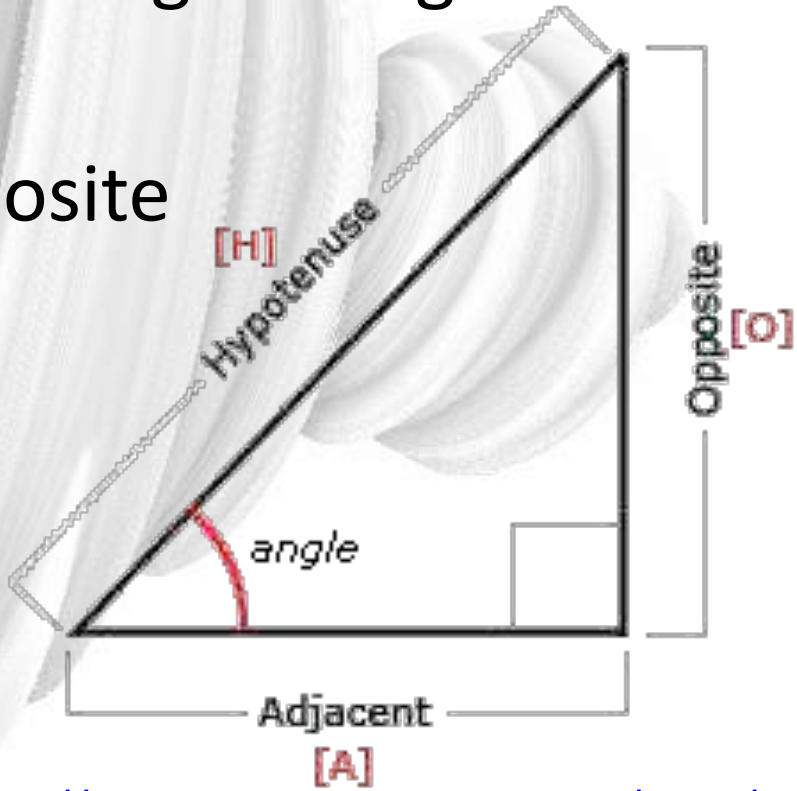
http://adrianboeing.blogspot.com/2011/01/xor-demoeffect-in-webgl.html

WebGL Demo Effect: Tunnel

Tan

- Given a right triangle with two sides (adjacent, opposite) we can find the angle using inverse tan
- Let x = adjacent, y = opposite

$$\tan A = \frac{\text{opposite}}{\text{adjacent}} = \frac{a}{b}.$$



http://adrianboeing.blogspot.com/2011/01/webgl-tunnel-effect-explained.html

WebGL Angle Visualization

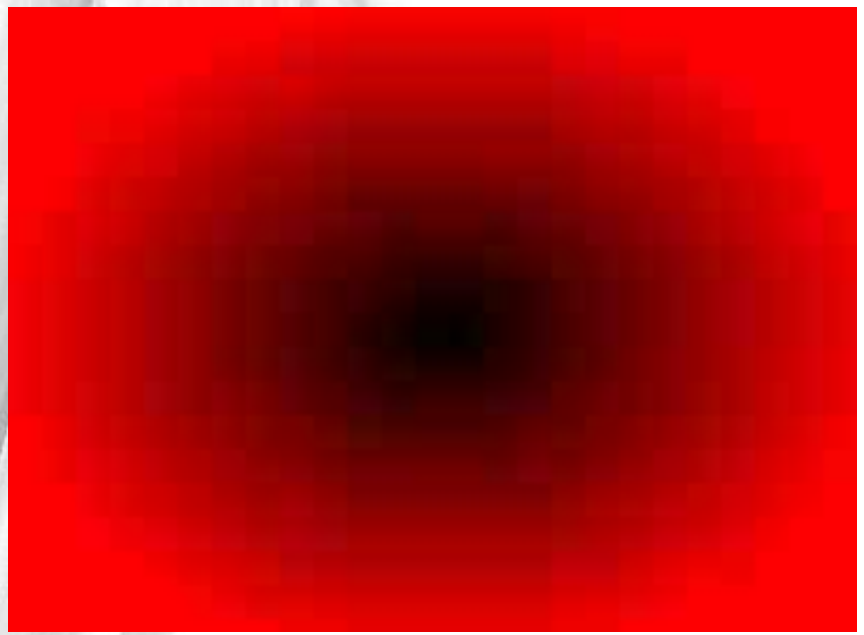
```
precision highp float;
uniform vec2 resolution;

void main(void) {
    vec2 p = -1.0 + 2.0 * gl_FragCoord.xy /
    resolution.xy;
    float a = atan(p.y,p.x);
    float result = a/(3.1416);
    gl_FragColor = vec4(result,0.0,0.0,1.0);
}
```

http://adrianboeing.blogspot.com/2011/01/webgl-tunnel-effect-explained.html

Remember the circle?

float r = sqrt(dot(p,p));

This would be a tunnel if it was 3d.



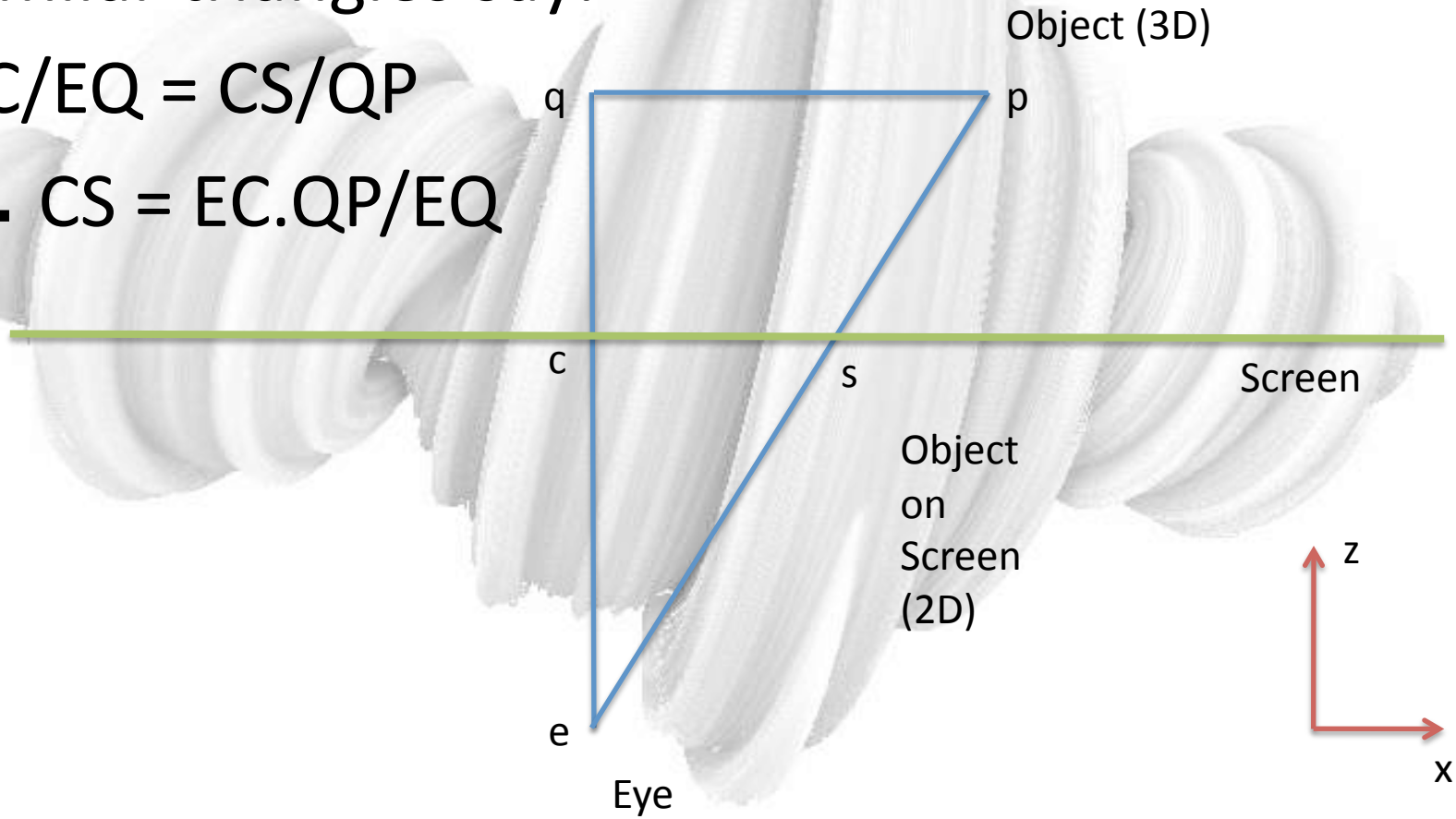
http://adrianboeing.blogspot.com/2011/01/webgl-tunnel-effect-explained.html

How to get from 2D to 3D

Similar triangles say:

EC/EQ = CS/QP

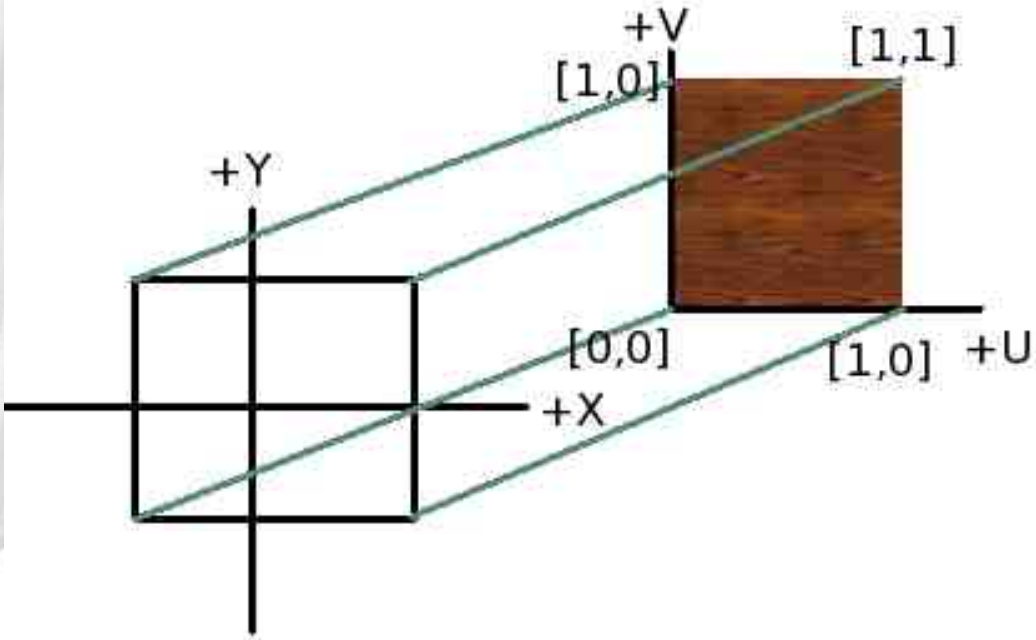
∴ CS = EC.QP/EQ



http://adrianboeing.blogspot.com/2011/01/webgl-tunnel-effect-explained.html

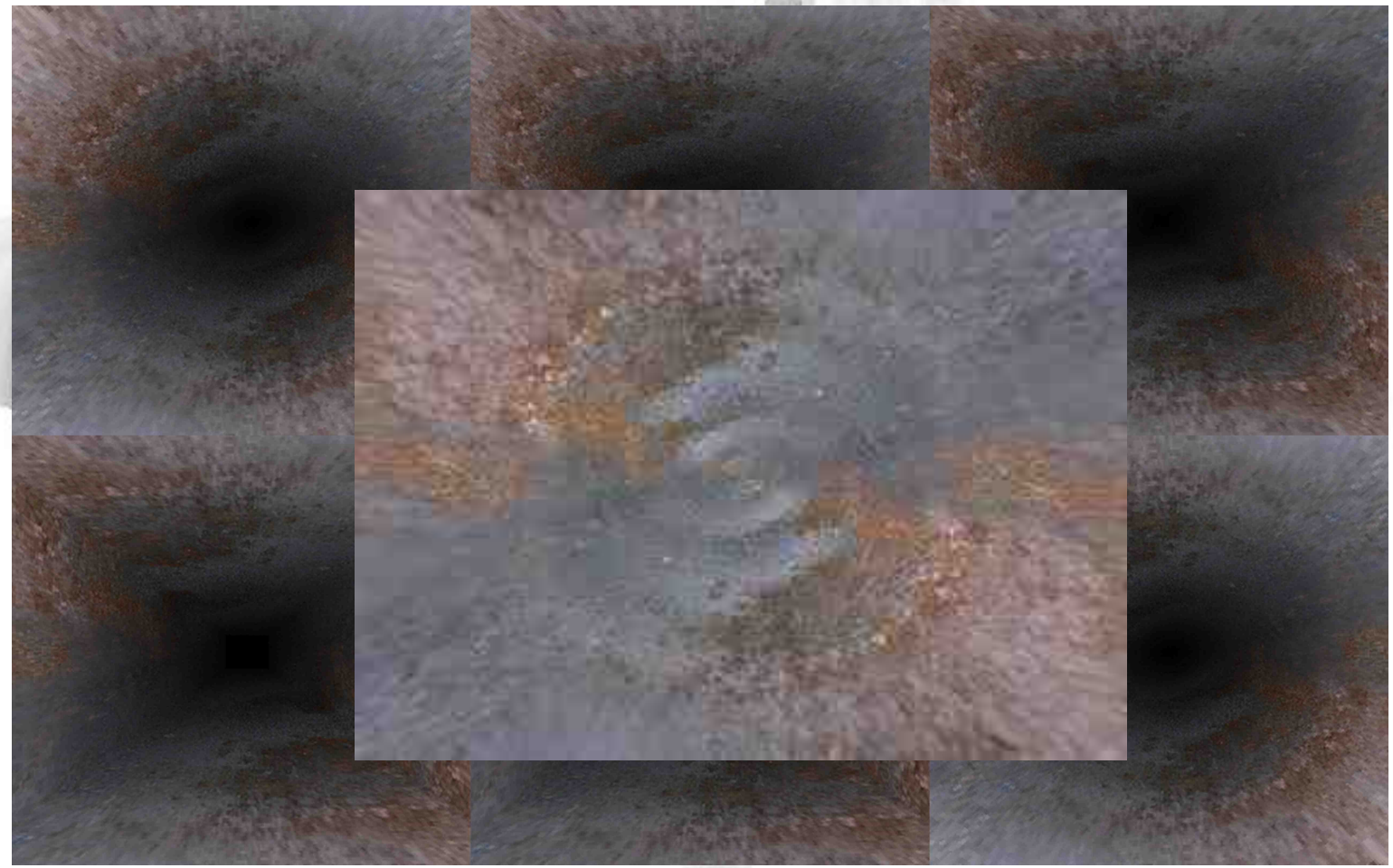
WebGL Texture Mapping

- uniform sampler2D tex;
- vec2 uv;
- vec3 col = texture2D (tex,uv).rgb;



http://adrianboeing.blogspot.com/2011/01/webgl-tunnel-effect-explained.html

WebGL Tunnel



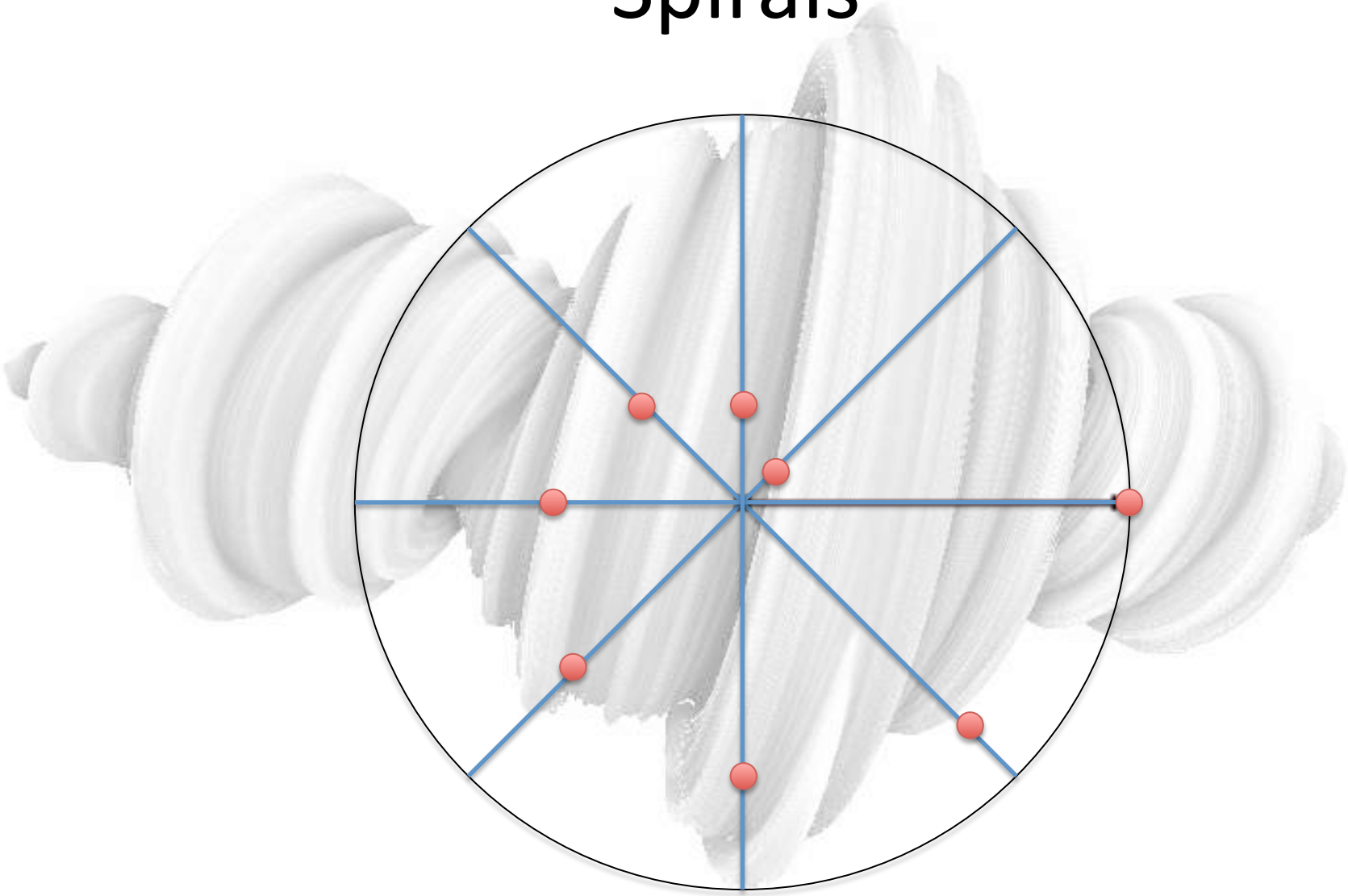
http://adrianboeing.blogspot.com/2011/01/webgl-tunnel-effect-explained.html

WebGL Tunnel

```
precision highp float;
uniform vec2 resolution;
uniform float time;
uniform sampler2D tex;

void main(void) {
vec2 p = -1.0 + 2.0 * gl_FragCoord.xy / resolution.xy;
vec2 uv;
float a = atan(p.y,p.x);
float r = sqrt(dot(p,p));
uv.x = 0.1/r + time;
uv.y = a/(3.1416);
vec3 col = texture2D(tex,uv).xyz;
gl_FragColor = vec4(col,1.0);
}
```

WebGL Demo Effect: Twist

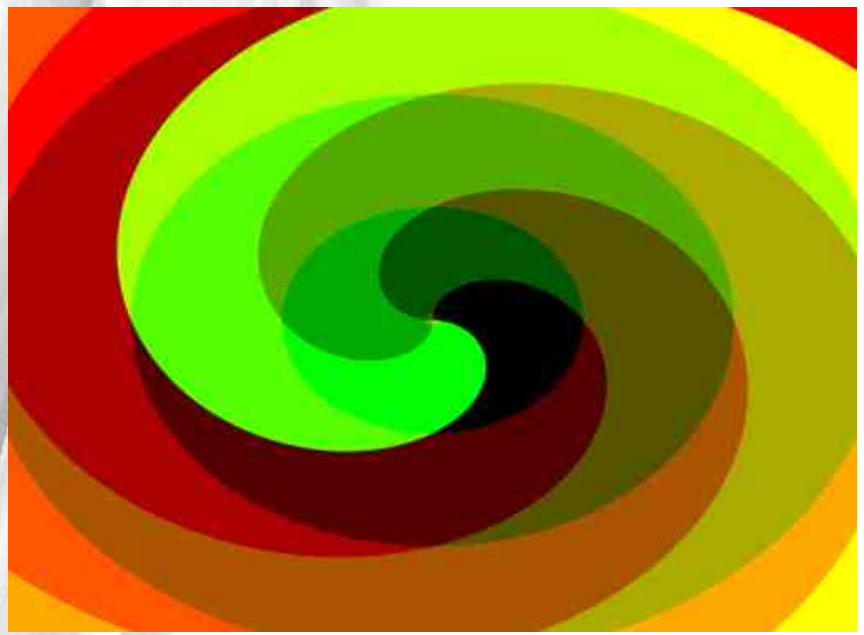
Spirals



http://adrianboeing.blogspot.com/2011/01/twist-effect-in-webgl.html

WebGL Twist

```
uniform vec2 resolution;
uniform sampler2D tex;

void main(void) {
vec2 p = -1.0 + 2.0 *
     gl_FragCoord.xy / resolution.xy;
vec2 uv;
float a = atan(p.y,p.x) / (2.0*3.1416);
float r = sqrt(dot(p,p)) / sqrt(2.0);
uv.x = r;
uv.y = a+r;
vec3 col = texture2D(tex,uv).xyz;
gl_FragColor = vec4(col,1.0);
}
```



http://adrianboeing.blogspot.com/2011/01/twist-effect-in-webgl.html

WebGL Demo Effect: Julia Fractal

Julia Fractal

- The term *fractal* was coined by Benoît Mandelbrot in 1975
- Gaston Julia (1893-1978) was a French mathematician whose work (published in 1918) inspired Mandelbrot

$$z_{n+1} = z_n^2 + c$$

Complex Numbers

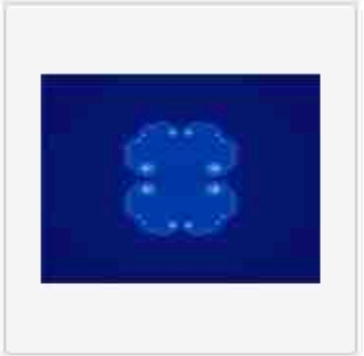
$(x + yi)(u + vi) = (xu - yv) + (xv + yu)i$

To render the fractal, we repeat $z_{n+1} = z_n^2 + c$ until a specified depth and render the number of iterations as a color

Julia Fractal

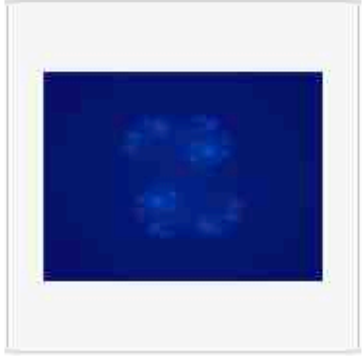
Filled Julia set for f_c, $c=1-\phi$ where ϕ is the golden ratio

Julia set for f_c, $c=(\phi-2)+(\phi-1)i$ =-0.4+0.6i

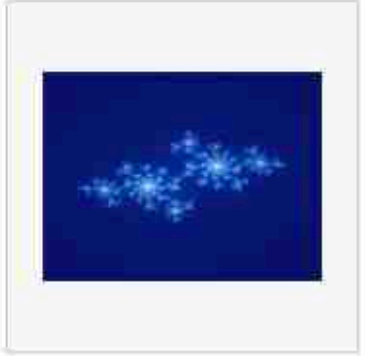


Julia set for f_c, c=0.285+0i

Julia set for f_c, c=0.285+0.01i



Julia set for f_c, c=0.45+0.1428i



Julia set for f_c, c=-0.70176-0.3842i

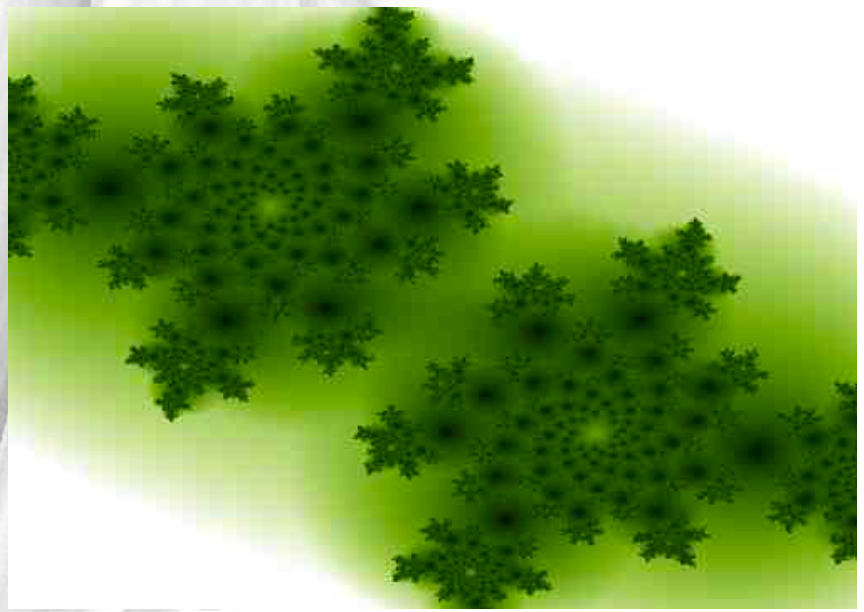
Julia set for f_c, c=-0.835-0.2321i

Julia set for f_c, c=-0.8+0.156i

http://en.wikipedia.org/wiki/Julia_set

WebGL Julia

```
uniform vec2 resolution;
uniform float time;

void main(void) {
   vec2 z = -1.0 + 2.0 * gl_FragCoord.xy /
      resolution.xy;
   vec2 cc = vec2(-0.4,0.6);
   float dmin = 1000.0;
   for( int i=0; i<64; i++ ) {
      z = cc + vec2( z.x*z.x - z.y*z.y, 2.0*z.x*z.y );
      float m2 = dot(z,z);
      if( m2>100.0 ) break;
      dmin=min(dmin,m2);
   }
   float c = sqrt(dmin);
   gl_FragColor = vec4(c,c,c,1.0);
}
```



That's it!

- Learn More!
- Slides & Tutorials on WebGL & Demoeffects:
- http://adrianboeing.blogspot.com/
- The Demoscene:
- http://pouet.net/ and http://scene.org/
- WebGL:
- http://learningwebgl.com/blog/ and http://www.iquilezles.org/apps/shadertoy/

Thank you!

Adrian Boeing