# Implicit Function Ray Tracing

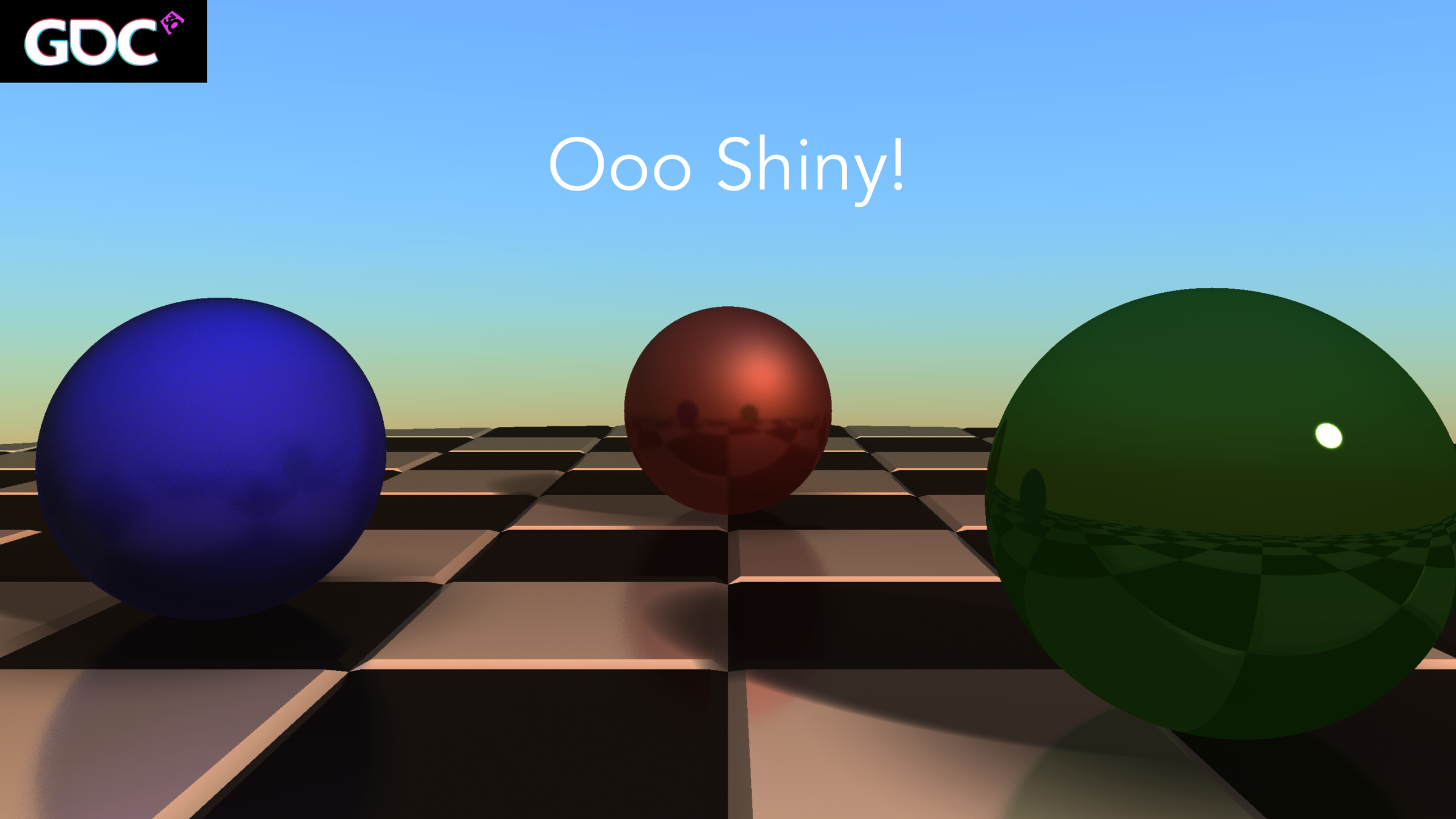
**Chris Birke**
Secant Astronomy

GAME DEVELOPERS CONFERENCE® March 14–18, 2016 · Expo: March 16–18, 2016 #GDC16

UBM

This talk is about rendering implicit functions, but it’s still totally cool if you don't know math or code.

This talk is about art, too.

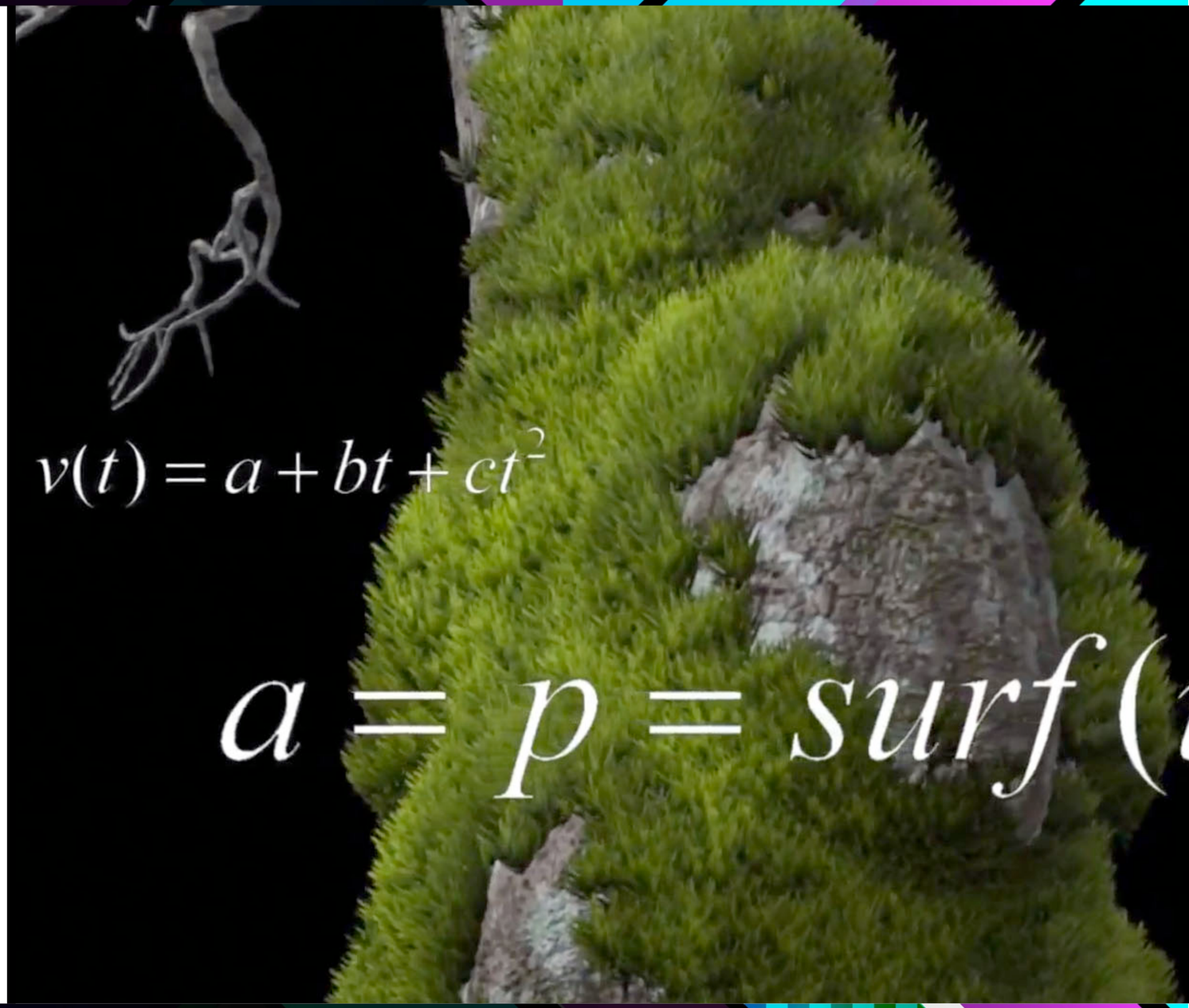

GDC
Ooo Shiny!

# A few people to thank.

# Iñigo Quílez

# Iñigo Quílez

aka iq



# Pablo Roman Andrioli

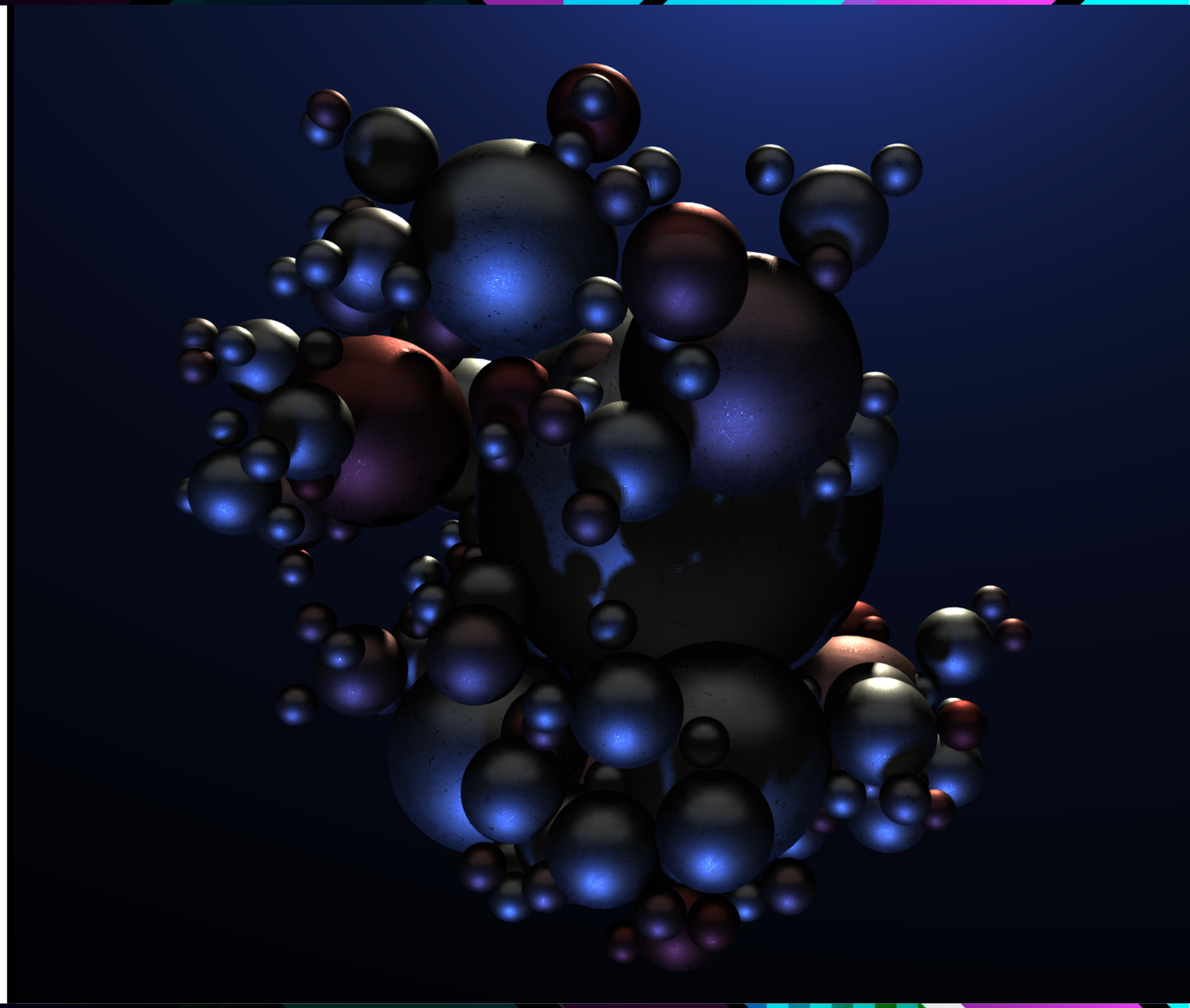
# Pablo Roman Andrioli

aka kali

# Ricardo Cabello

# Ricardo Cabello

aka Mr.doob



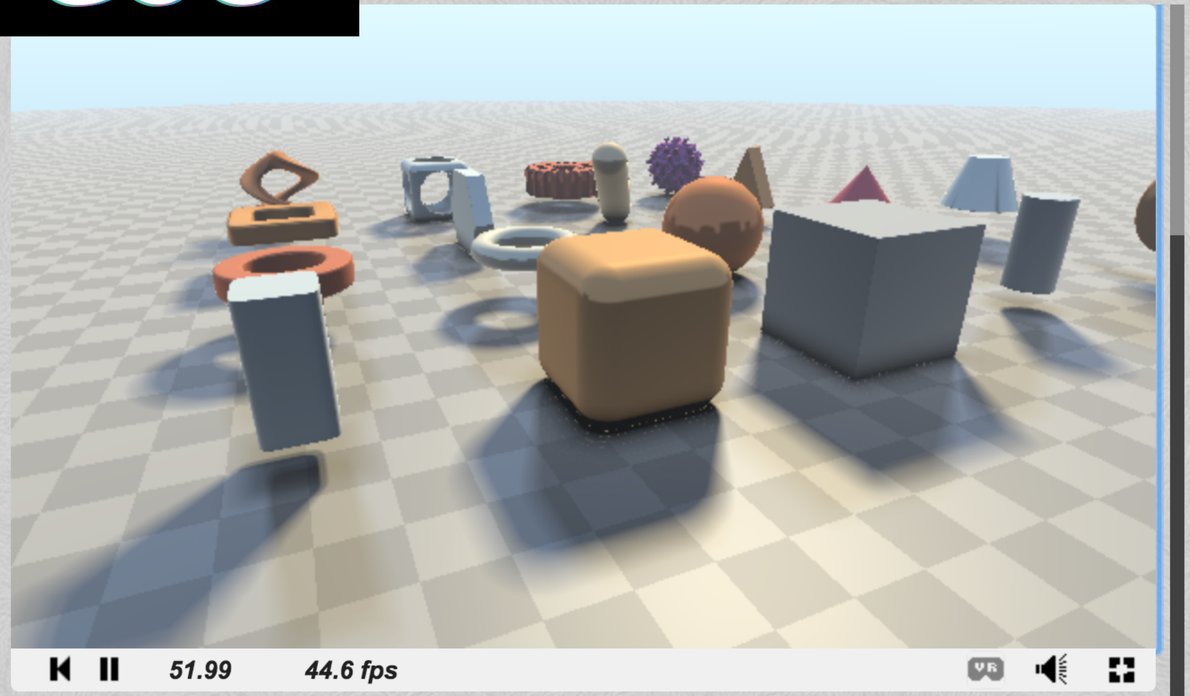
# A few places to thank.

mpiled successfully

fullscreen gallery

```glsl
#ifdef GL_ES
precision mediump float;
#endif

#extension GL_OES_standard_derivatives : enable

uniform float time;
uniform vec2 mouse;
uniform vec2 resolution;

void main( void ) {

	vec2 position = ( gl_FragCoord.xy / resolution.xy ) + mouse / 4.0;

	float color = 0.0;
	color += sin( position.x * cos( time / 15.0 ) * 80.0 ) + cos( position.y * cos( time / 15.0 ) * 10.0 );
	color += sin( position.y * sin( time / 10.0 ) * 40.0 ) + cos( position.x * sin( time / 25.0 ) * 40.0 );
	color += sin( position.x * sin( time / 5.0 ) * 10.0 ) + sin( position.y * sin( time / 35.0 ) * 80.0 );
	color *= sin( time / 10.0 ) * 0.5;

	gl_FragColor = vec4( vec3( color, color * 0.5, sin( color + time / 3.0 ) * 0.75 ), 1.0 );

}
```

# GLSL Sandbox



51.99 44.6 fps

# Raymarching - Primitives

58441 319 40

Uploaded by **iq** in 2013-Mar-25

Tags: **procedural**, **3d**, **raymarching**, **distancefields**, **primitives**

A set of primitives and combination functions, for reference. More info here:
**http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm**

Image

Shader Inputs

```
// Created by inigo quilez - iq/2013
// License Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

// A list of usefull distance function to simple primitives, and an example on how to
// do some interesting boolean operations, repetition and displacement.
//
// More info here: http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

float sdPlane( vec3 p )
{
    return p.y;
}

float sdSphere( vec3 p, float s )
{
    return length(p)-s;
}

float sdBox( vec3 p, vec3 b )
{
  vec3 d = abs(p) - b;
  return min(max(d.x,max(d.y,d.z)),0.0) + length(max(d,0.0));
}

float sdEllipsoid( in vec3 p, in vec3 r )
{
    return (length( p/r ) - 1.0) * min(min(r.x,r.y),r.z);
}

float udRoundBox( vec3 p, vec3 b, float r )
{
  return length(max(abs(p)-b,0.0))-r;
}

float sdTorus( vec3 p, vec2 t )
{
  return length( vec2(length(p.xz)-t.x,p.y) )-t.y;
}
```

6757 chars

Comments

**Sign in** to post a comment.

**hackthechad**, 2016-Feb-21
Awesome reference IQ--learning a lot! Quick question though. What are the bac and dom variables in your lighting calculations?

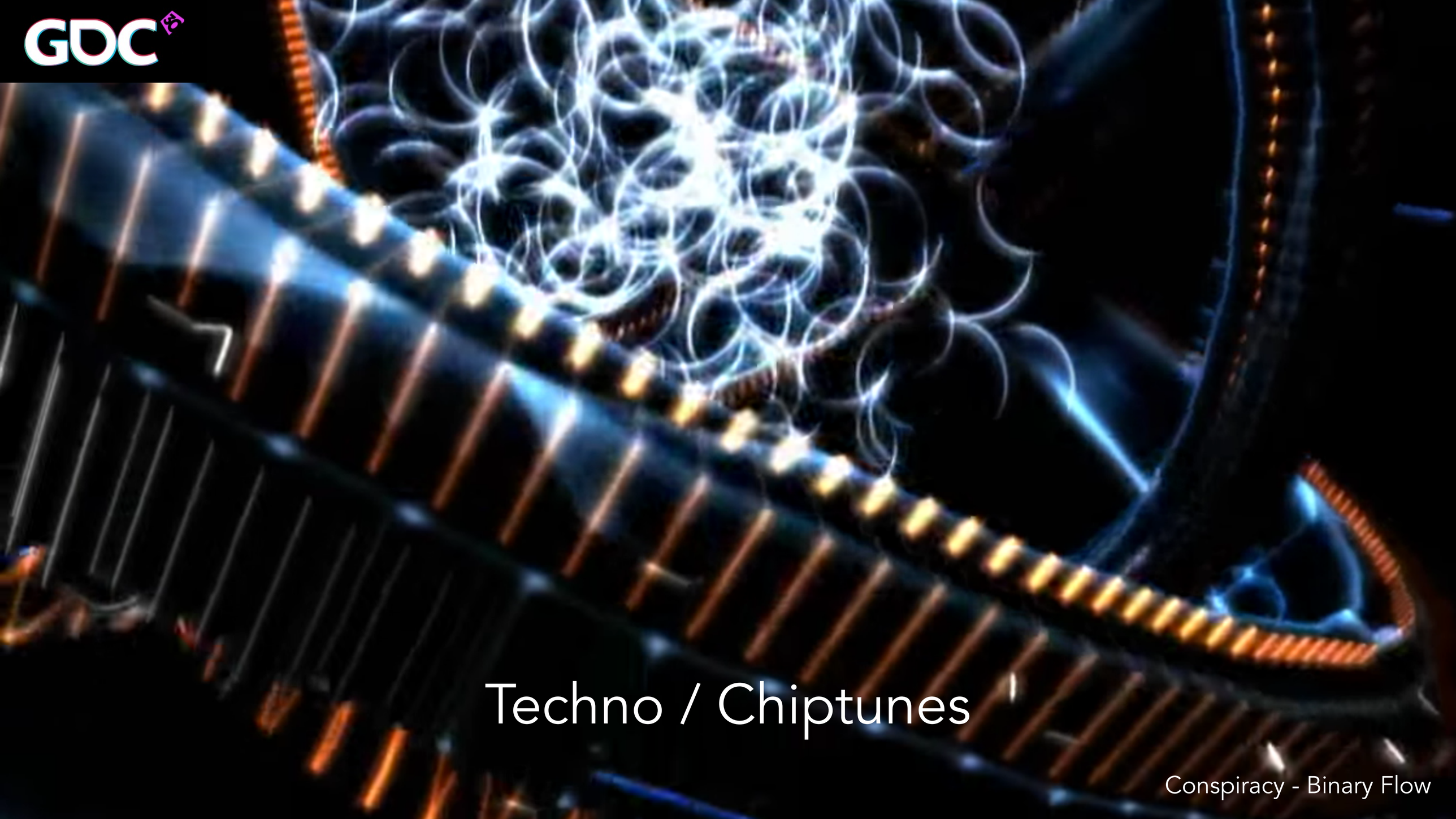
```
float bac = clamp( dot( nor, normalize(vec3(-lig.x,0.0,-lig.z))), 0.0, 1.0 )*clamp( 1.0-pos.y,0.0,1.0);
float dom = smoothstep( -0.1, 0.1, ref.y );
```

**Doublefresh**, 2016-Feb-13
What does the 'c' parameter in sdCone do?

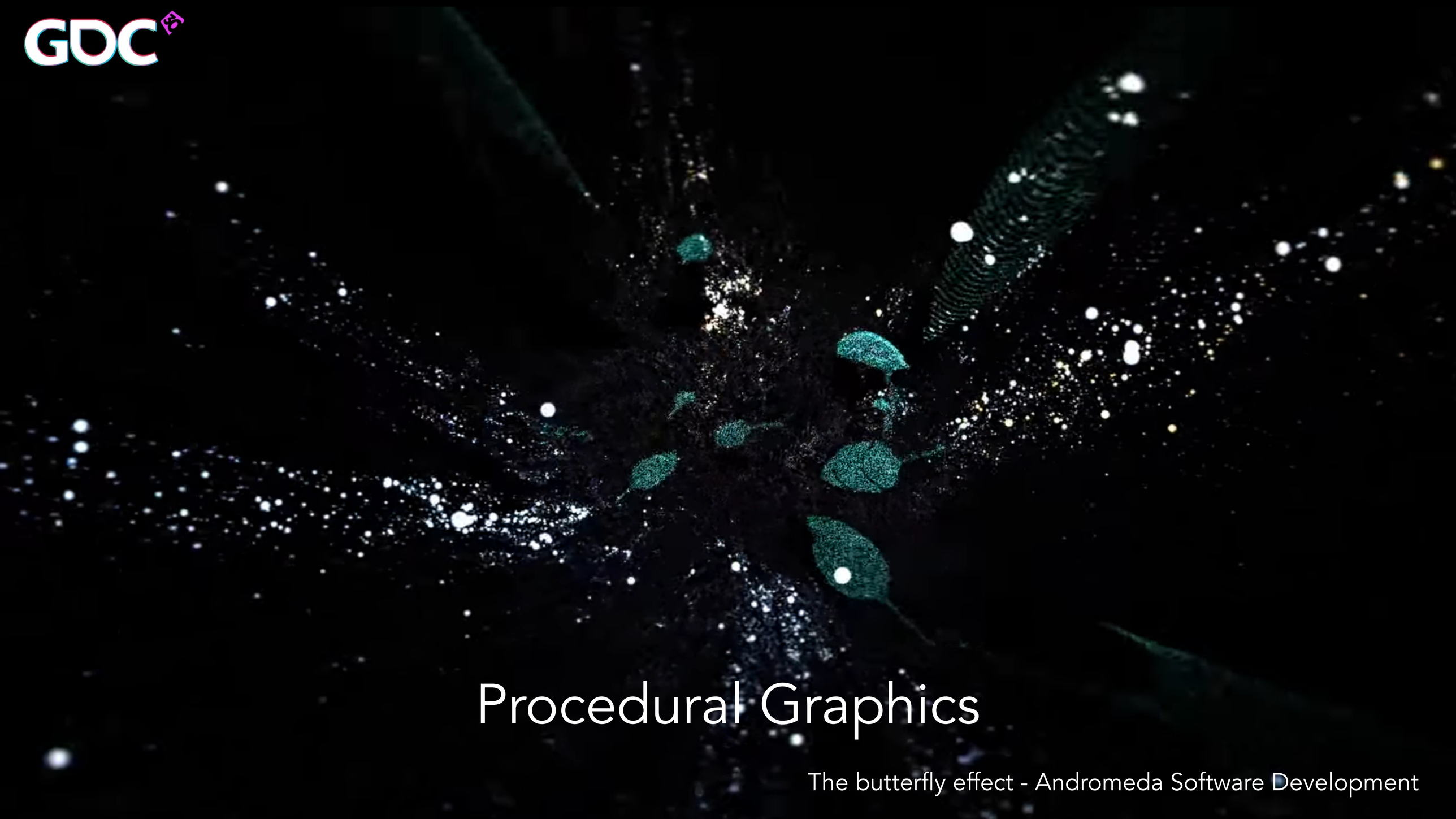
iChannel0 iChannel1 iChannel2 iChannel3

# Shadertoy

GDC
Quiet Steps by keyboarders


GDC
Techno / Chiptunes
Conspiracy - Binary Flow

Many Cubes
Digital Dynamite - Tilescapes



# Procedural Graphics

The butterfly effect - Andromeda Software Development

GDC
DEALER QUALITY SOFTWARE
TRIAD
JOINING, HIDING, LEAVING, PUBLIC PERFORM

FAIRLIGHT PRESENTS

FULL CONTACT FROM TEAM 17

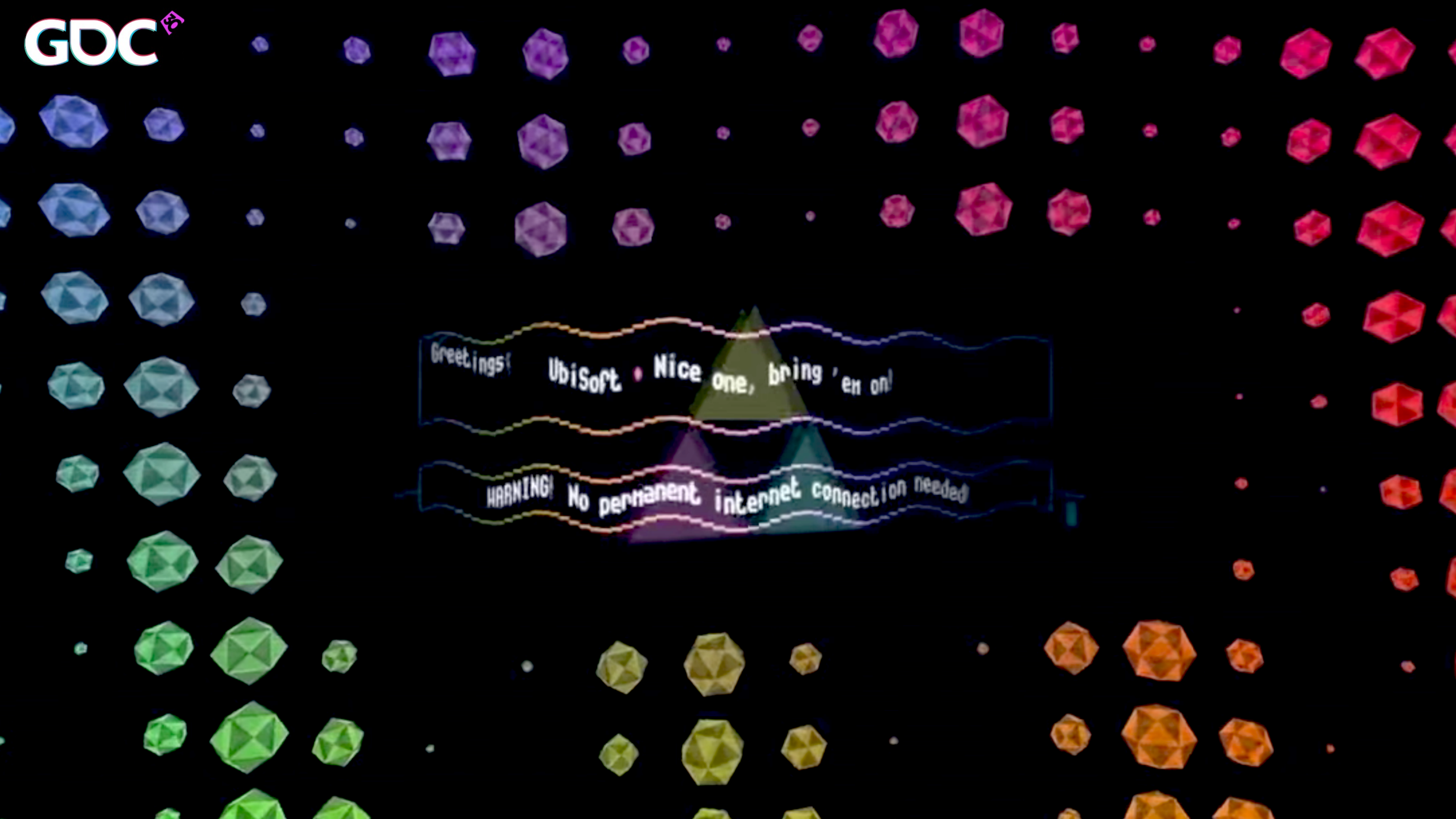
CRACKED BY GASTON

SUPPLIED BY DRONE NO.5

WRITE US NOW :

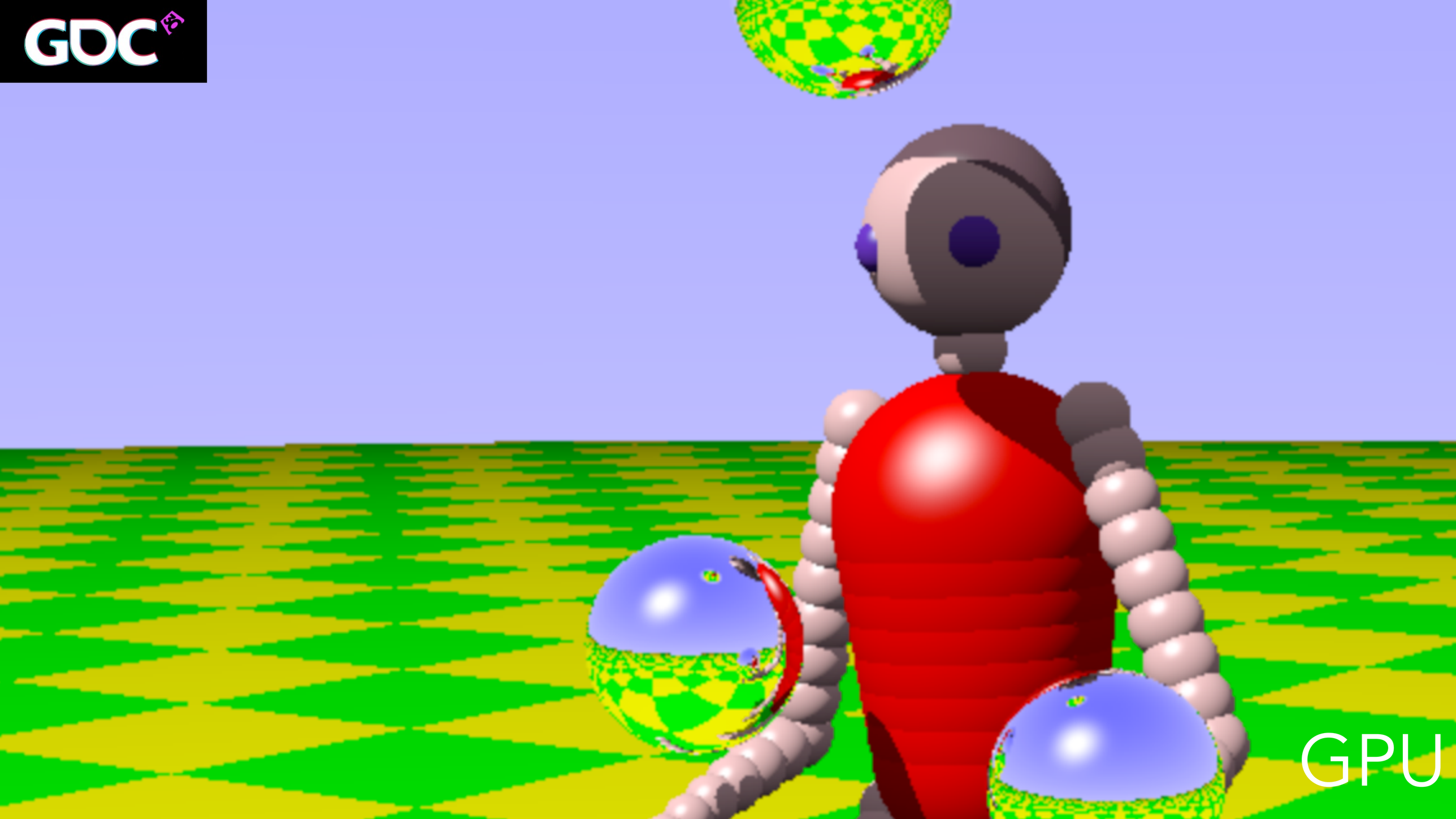
PO BOX 6, 236 00 HOLLVIKEN, SWEDEN

GDC
REZ


Greetings! UbiSoft • Nice one, bring 'em on!
WARNING! No permanent internet connection needed!

GDC
C64

Amiga


GPU

Photon - Fairlight

Intrinsic Gravity - Still

GDC
Assembly 2015

# Demoscene

Many Cubes

Much Techno

C64/Amiga/GPU

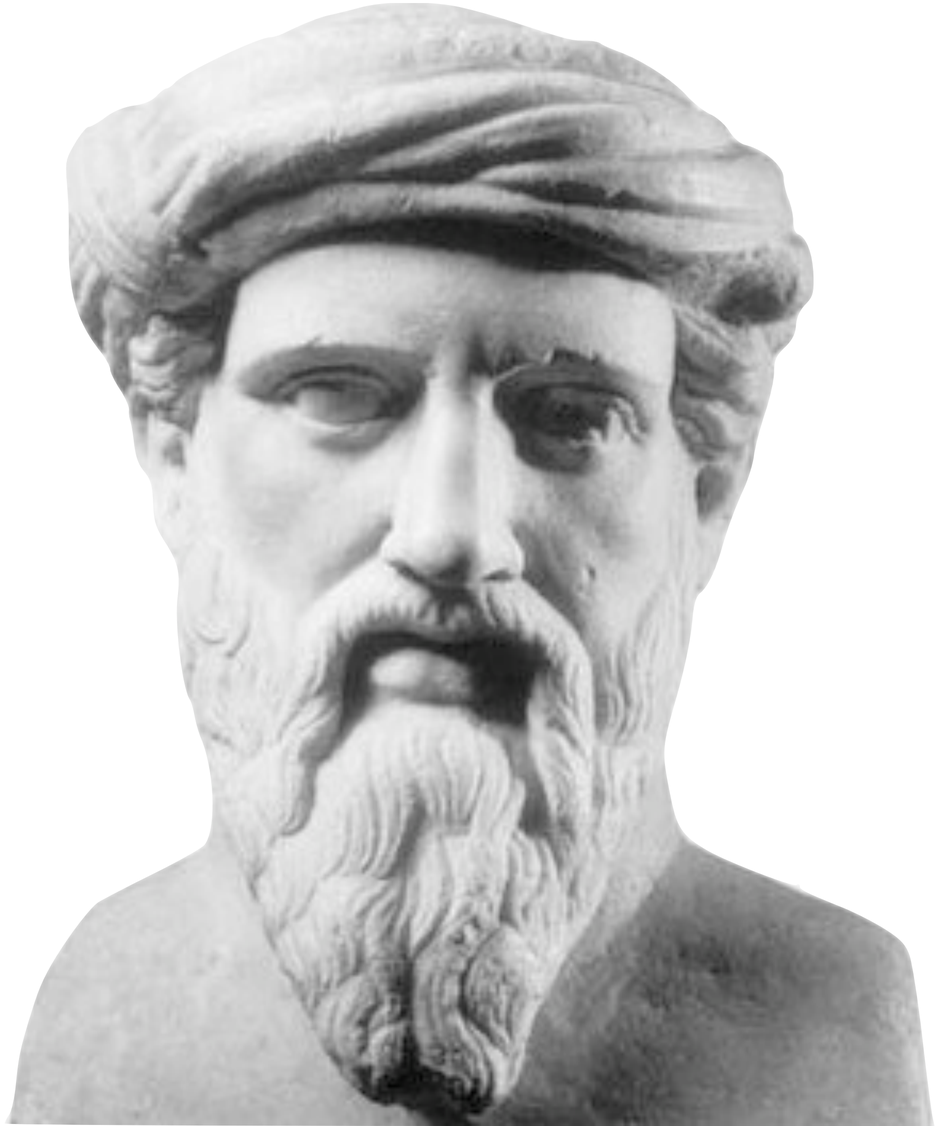
Procedural

Still Relevant

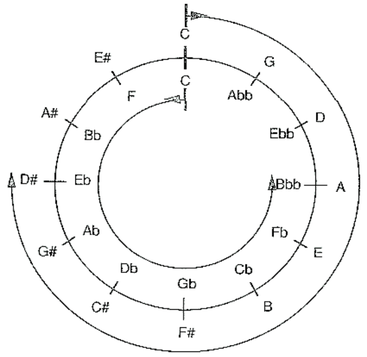
SF demoscene

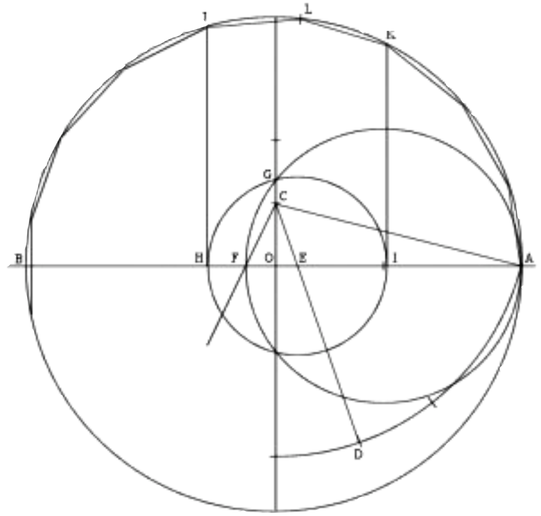
info goes here….

# Πυθαγόρας

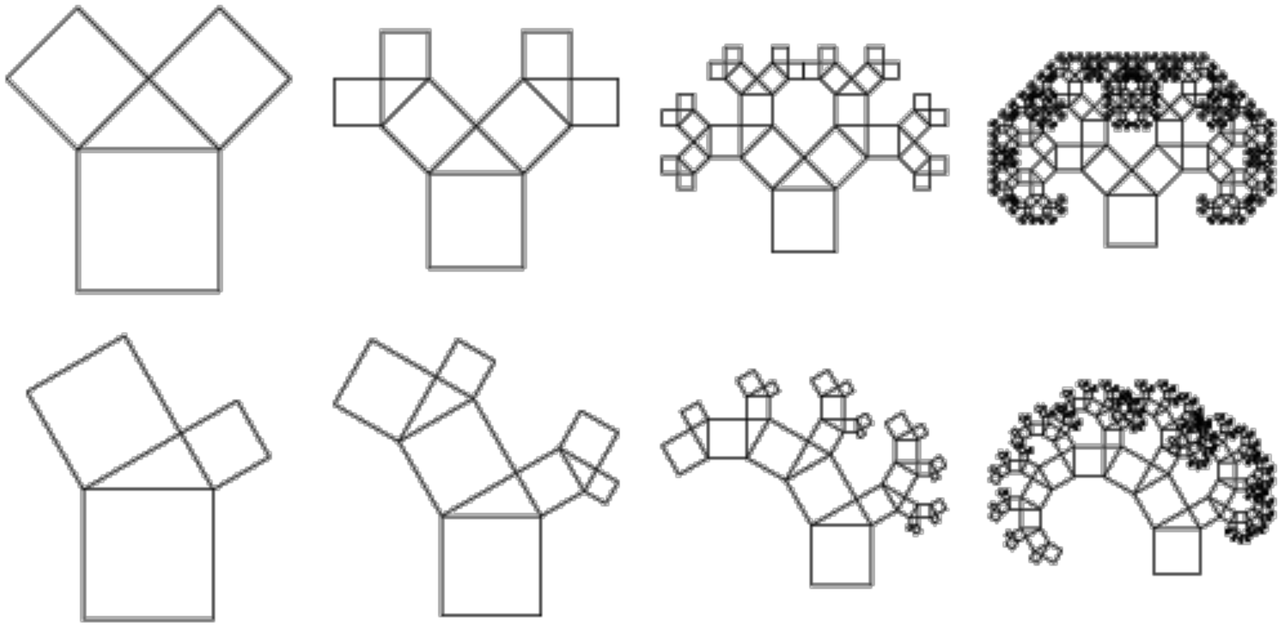


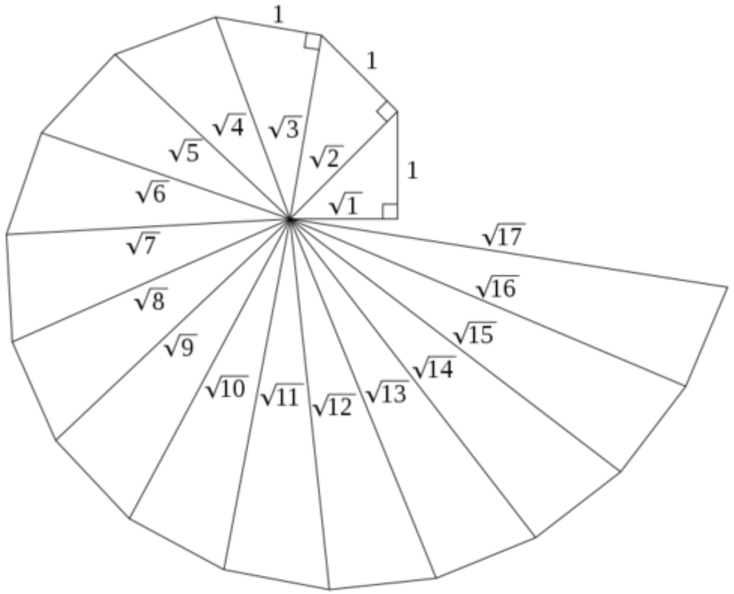
# Pythagoras




Discovered tons of crazy stuff.

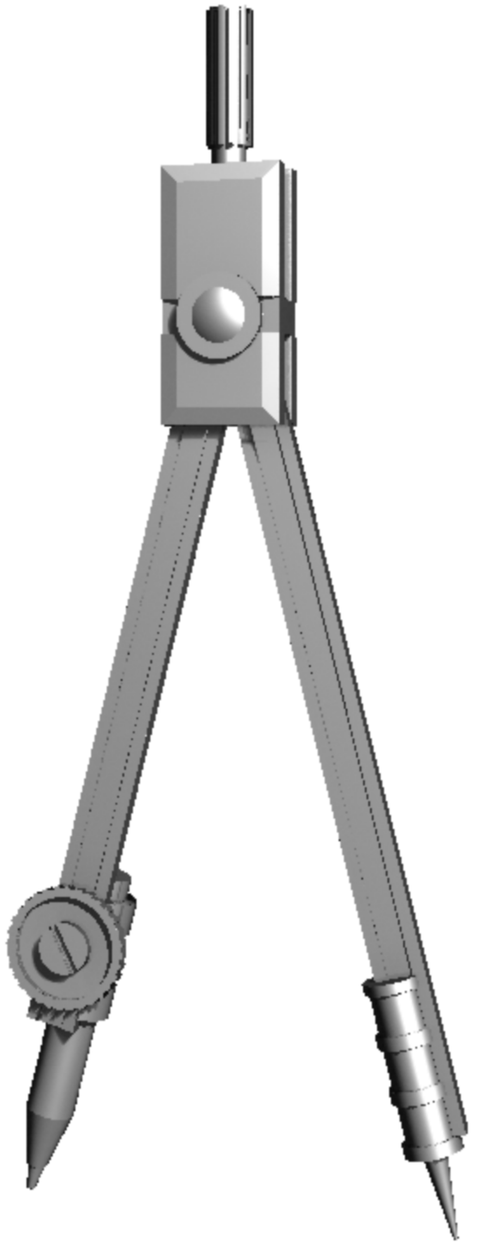




2500 years ago.

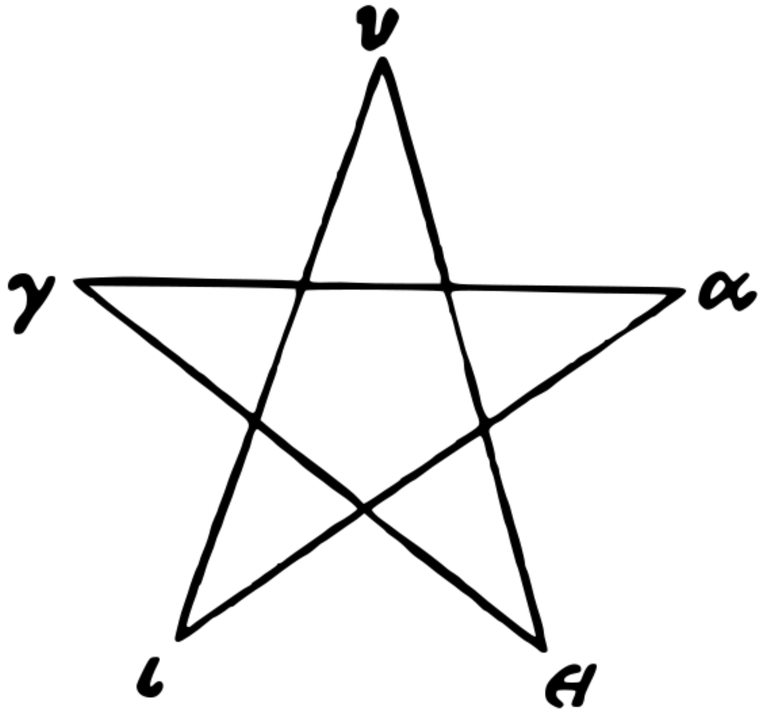
# Pythagoras

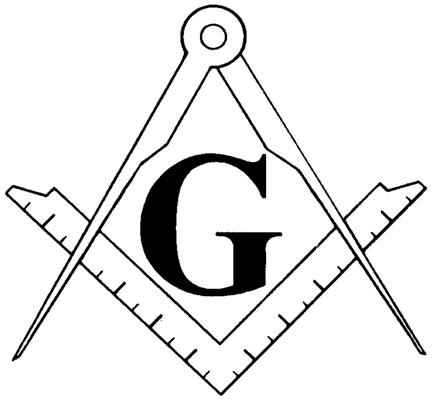
Straight edge (and compass.)

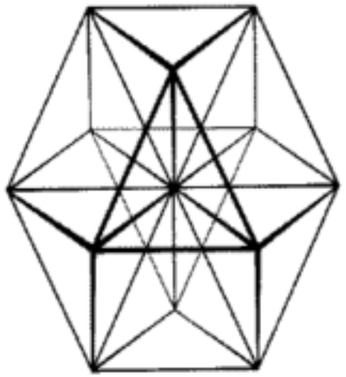


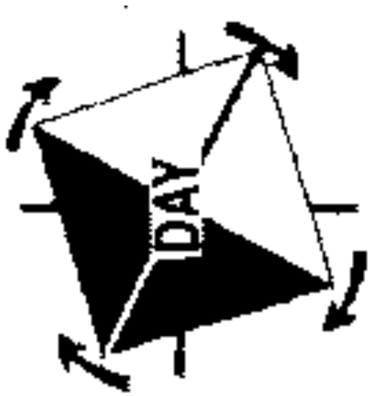
# Pythagoras

Spawned numerous math cults.

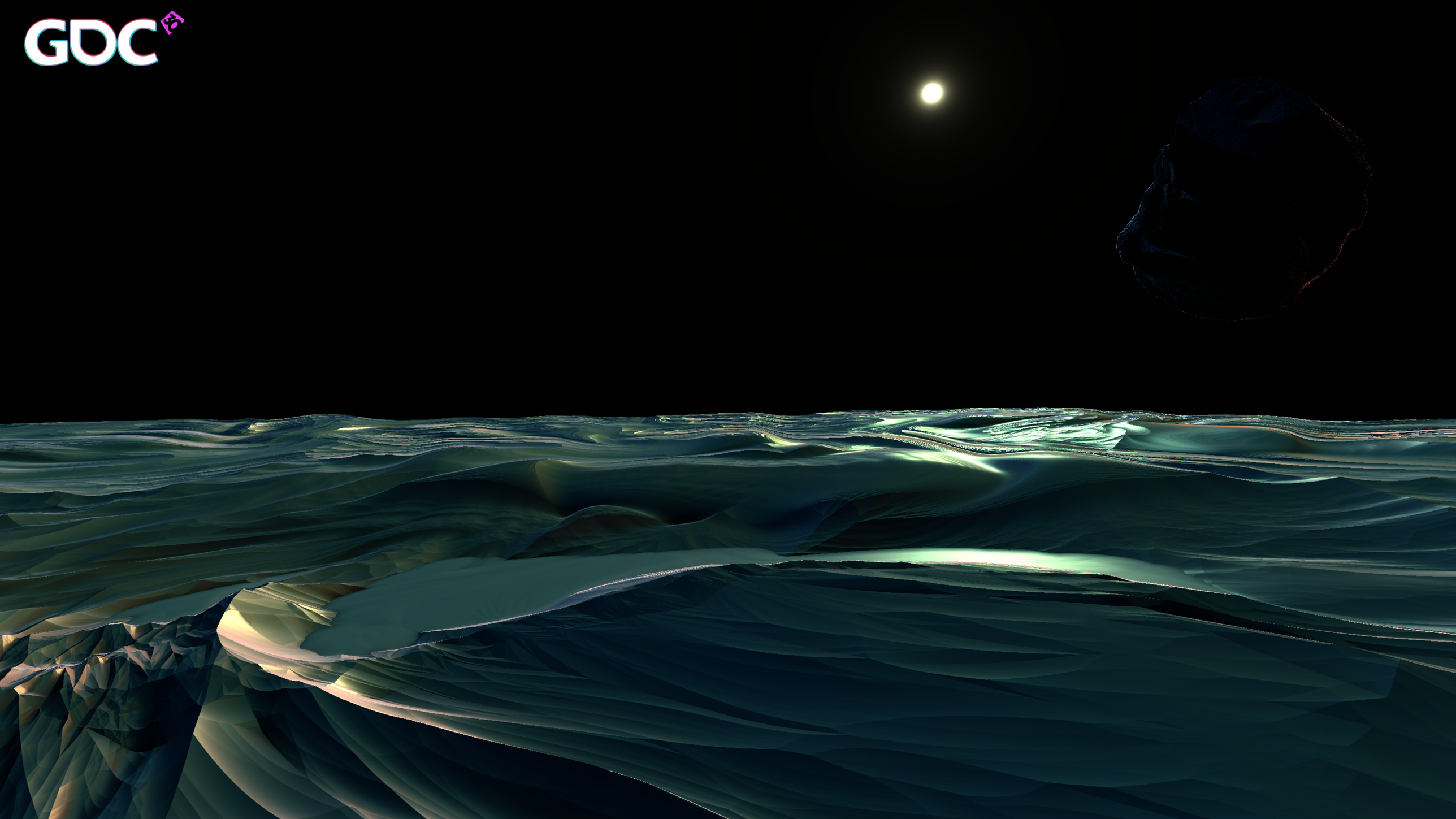


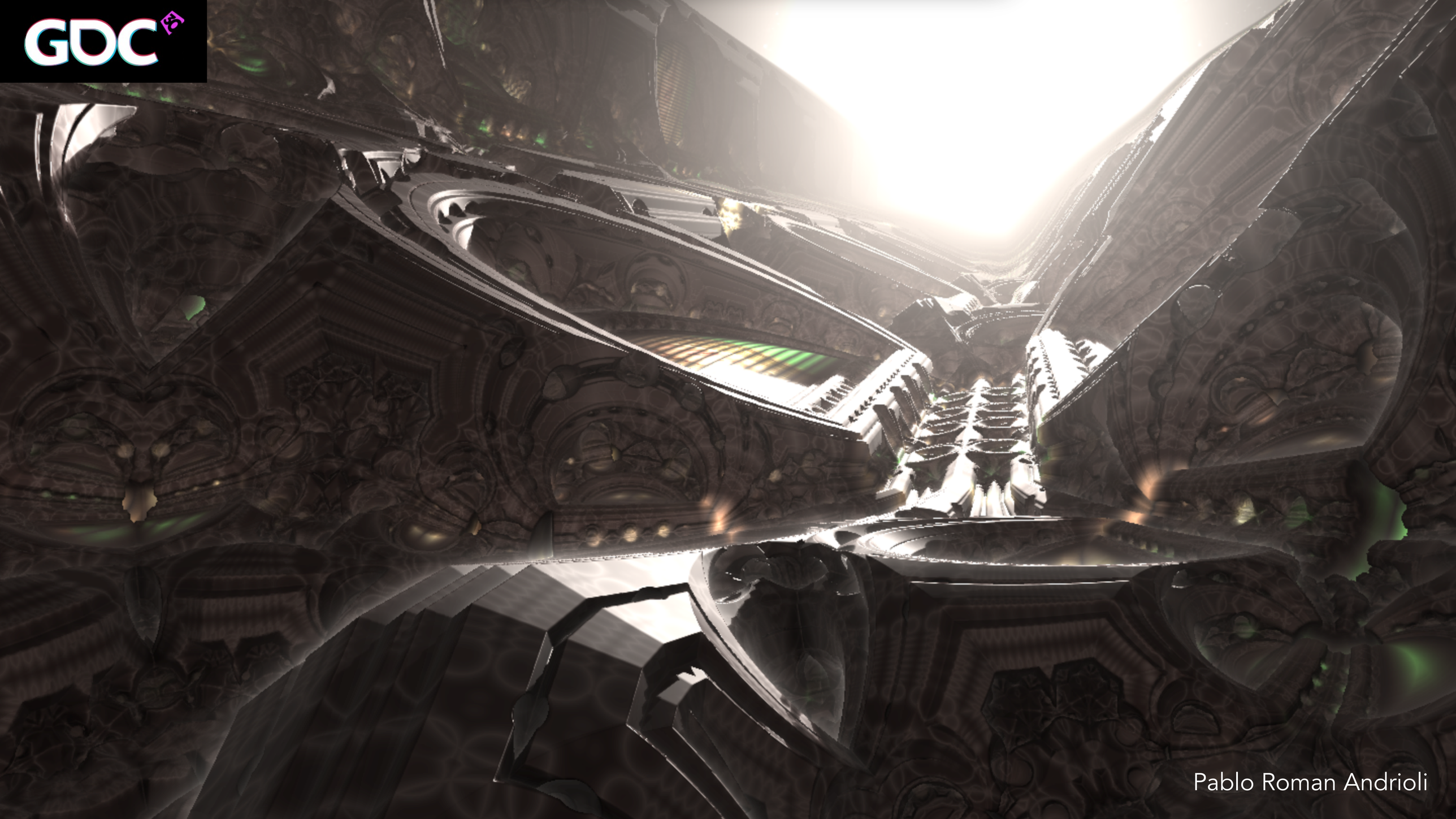



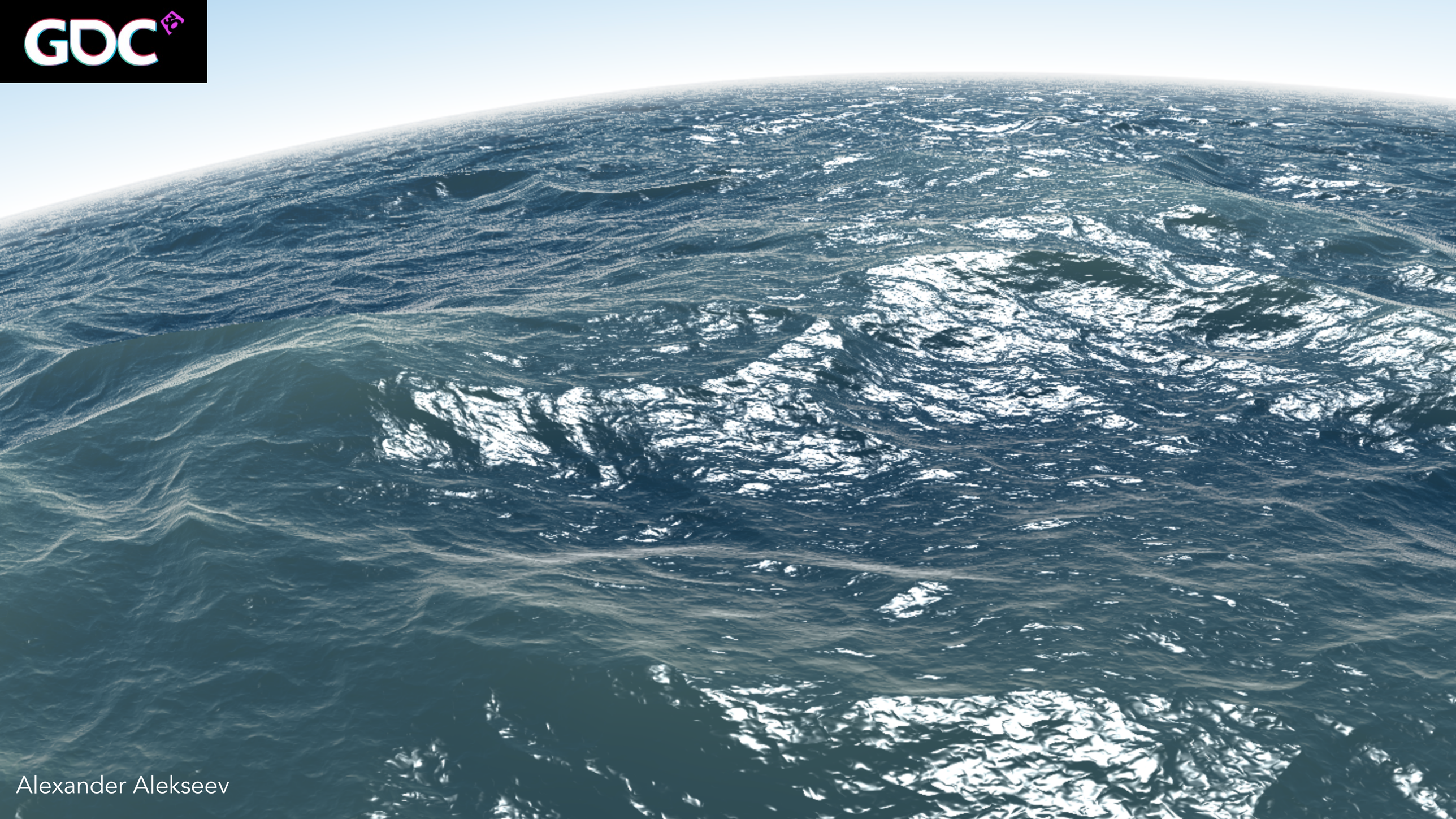





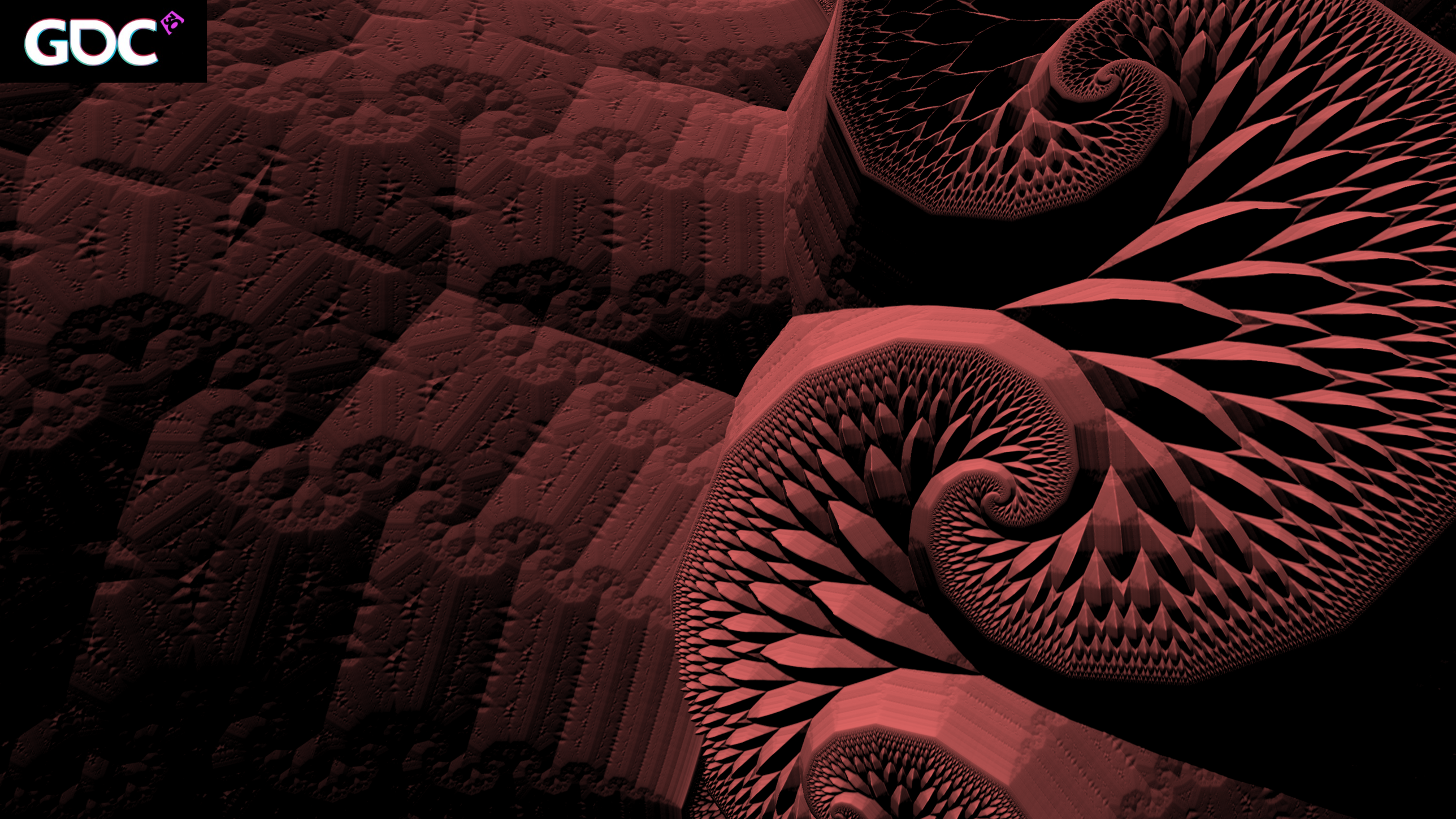
The universe is math.




GDC
Pablo Roman Andrioli



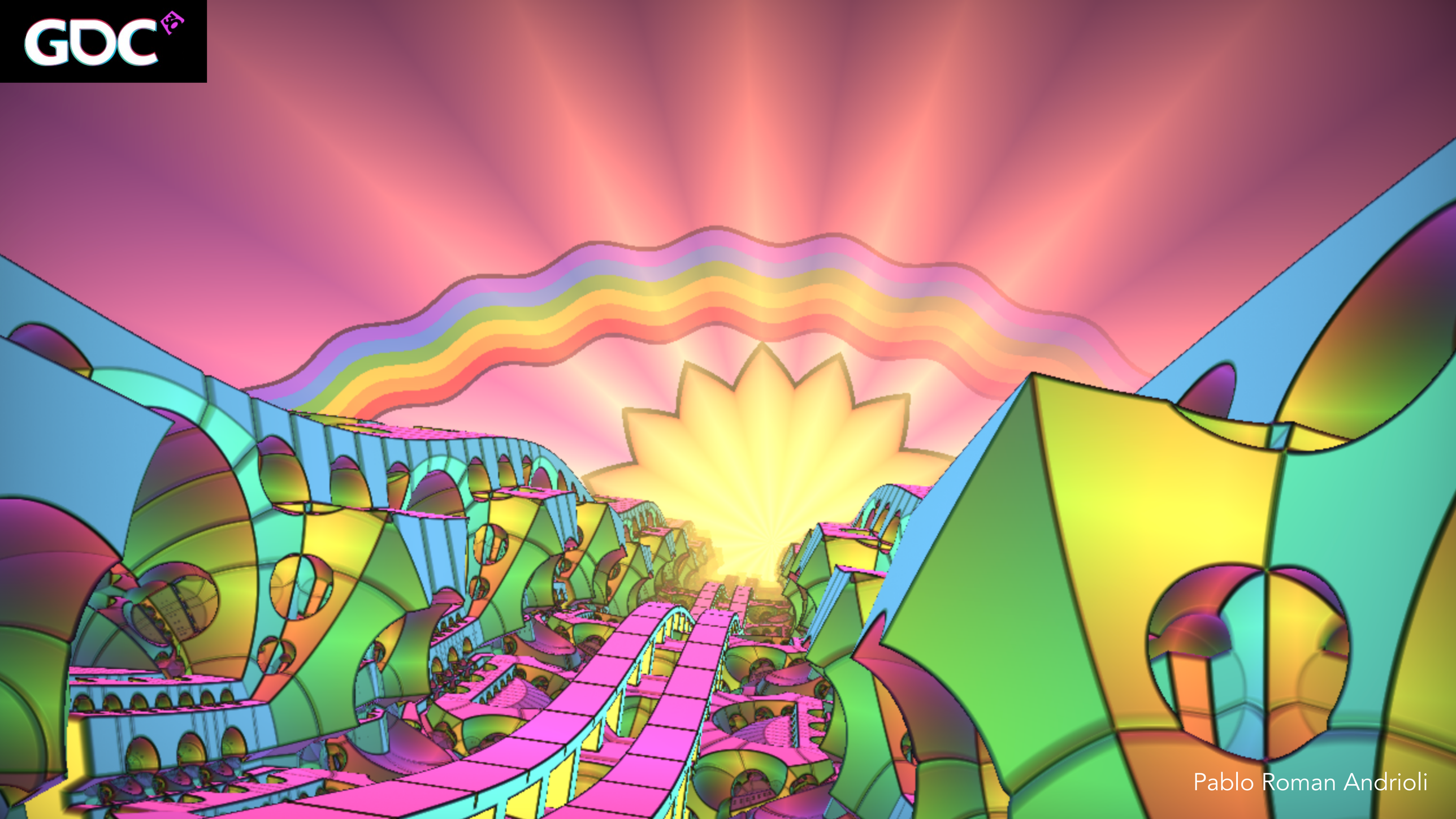
Alexander Alekseev

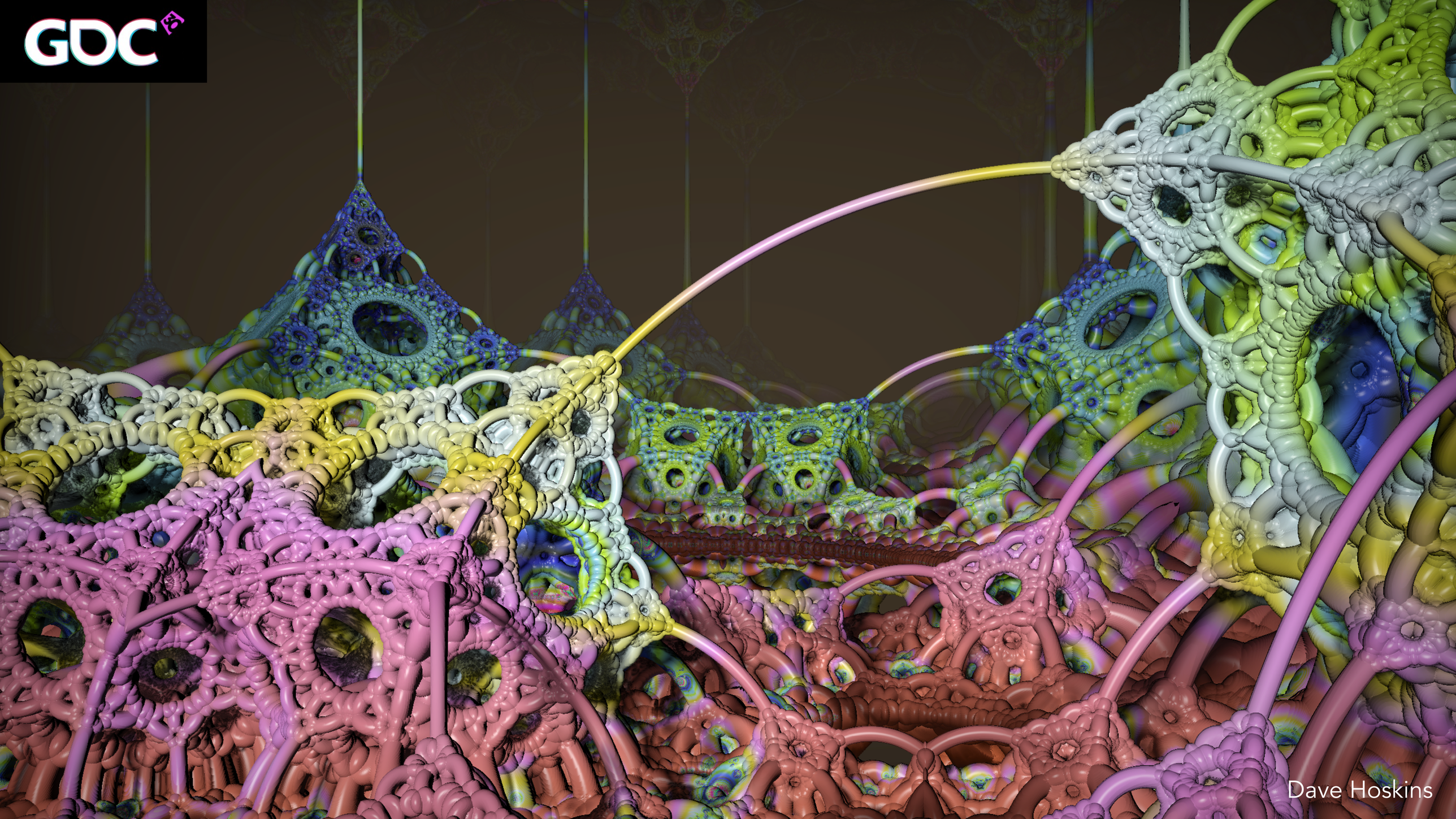


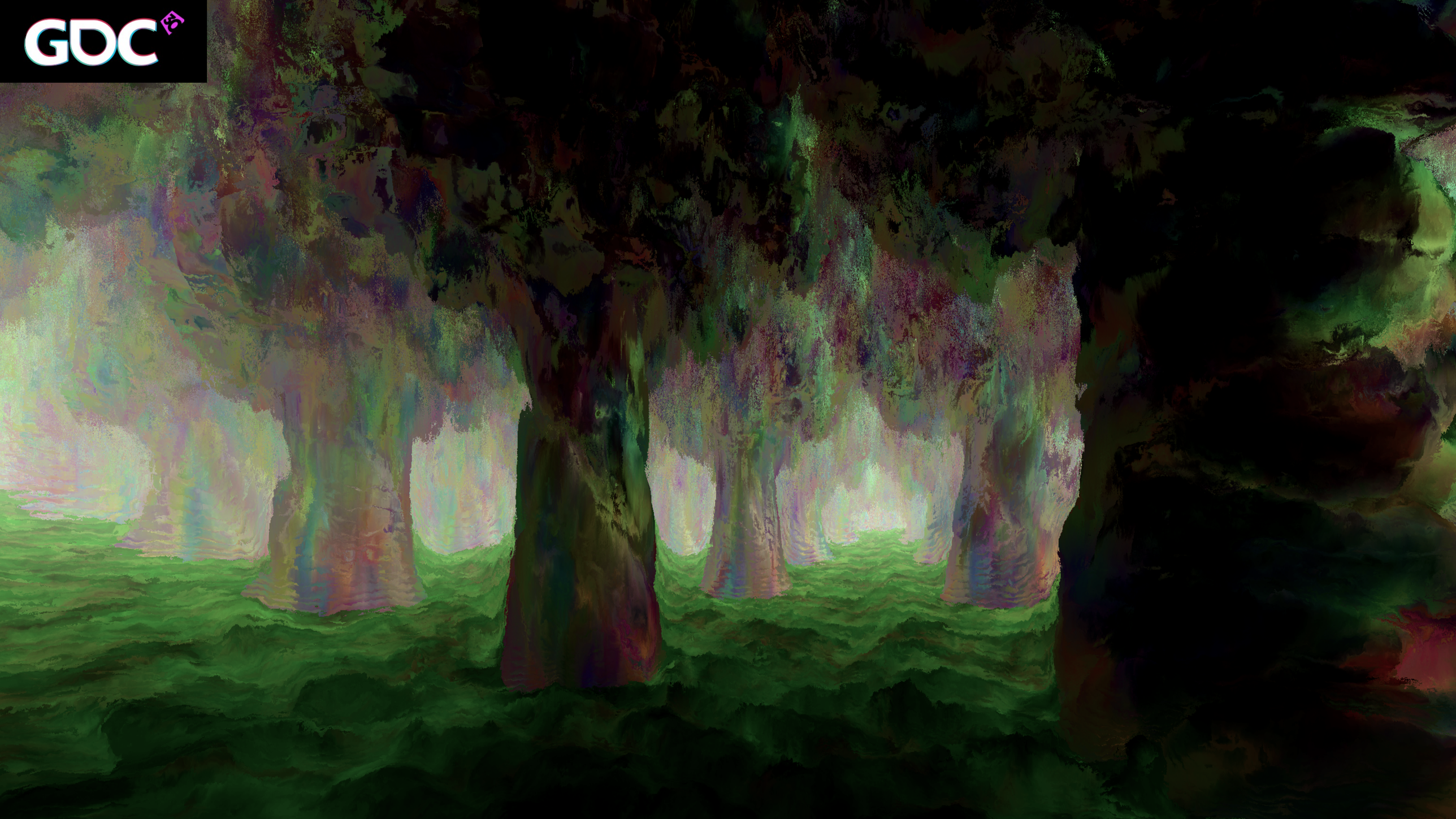
GDC
Dave Hoskins

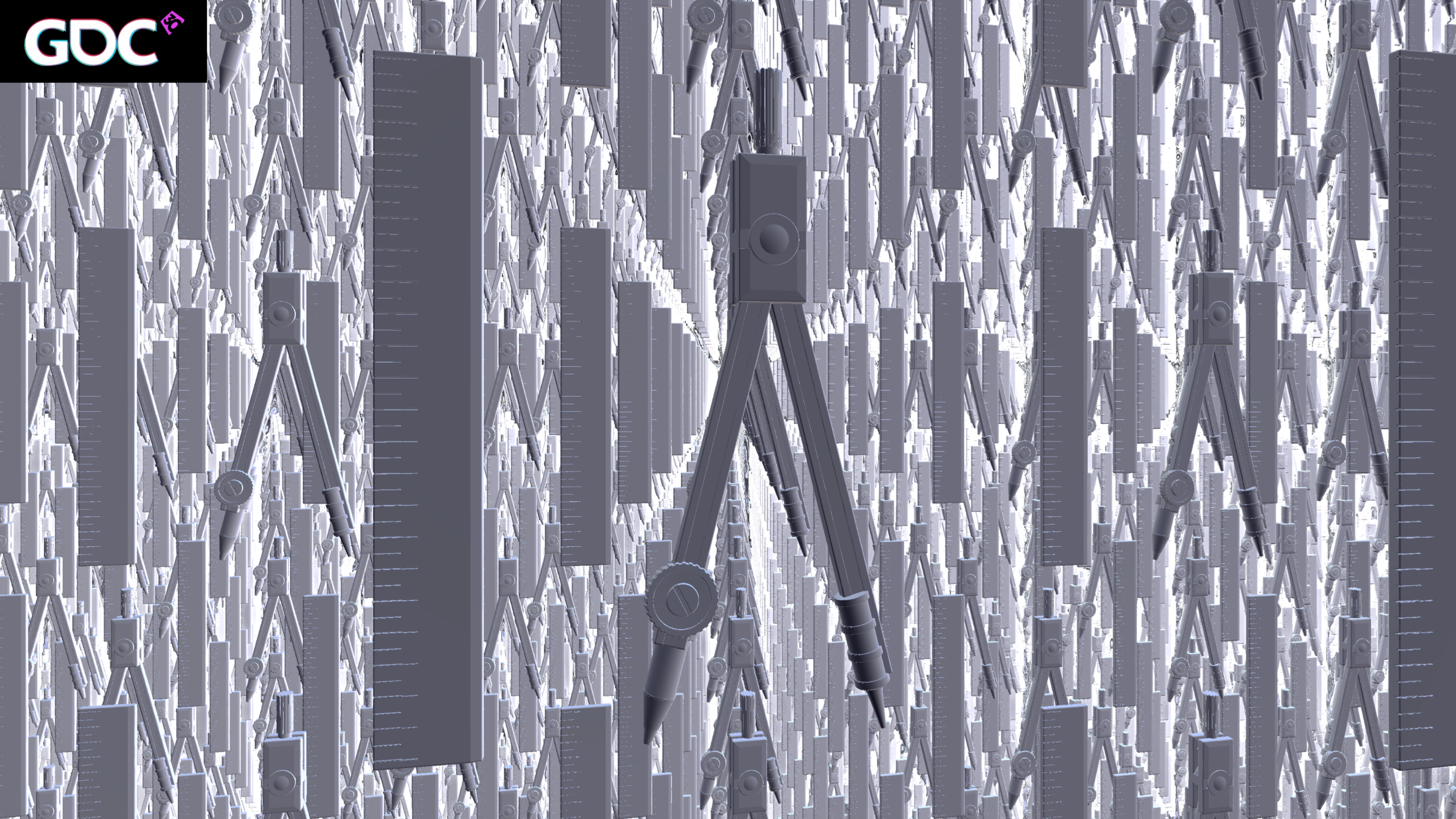
GDC
Reinder Nijhoff

GDC
Iñigo Quílez


GDC
Pablo Roman Andrioli


GDC
Dave Hoskins


GDC


GDC

If the universe is math…

Raytracing is a nice way to visit.

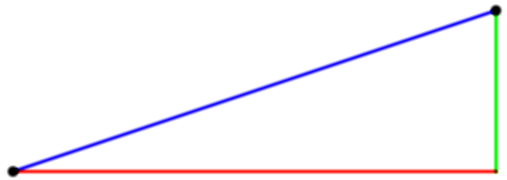
This is the point of the talk.

Actually, there are two points.

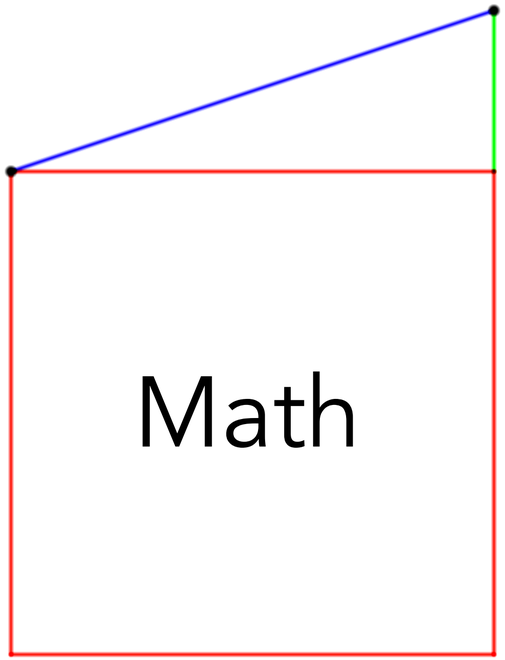
How far apart are they?

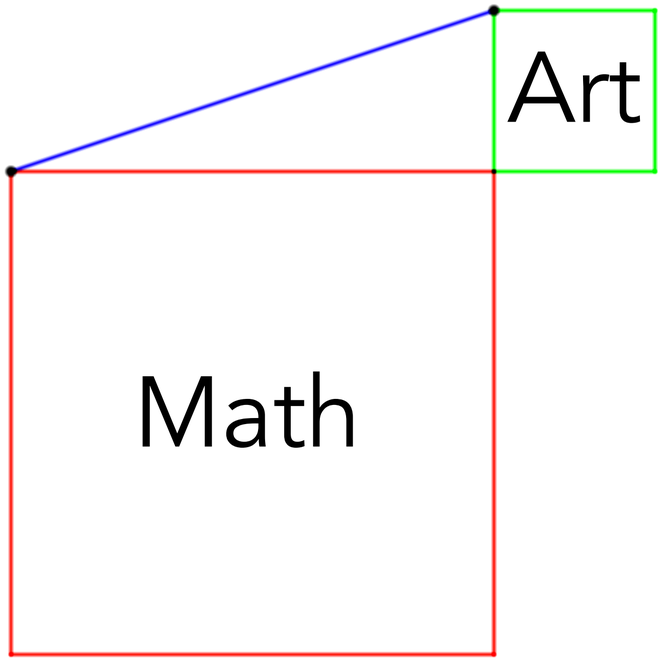
Time for math.

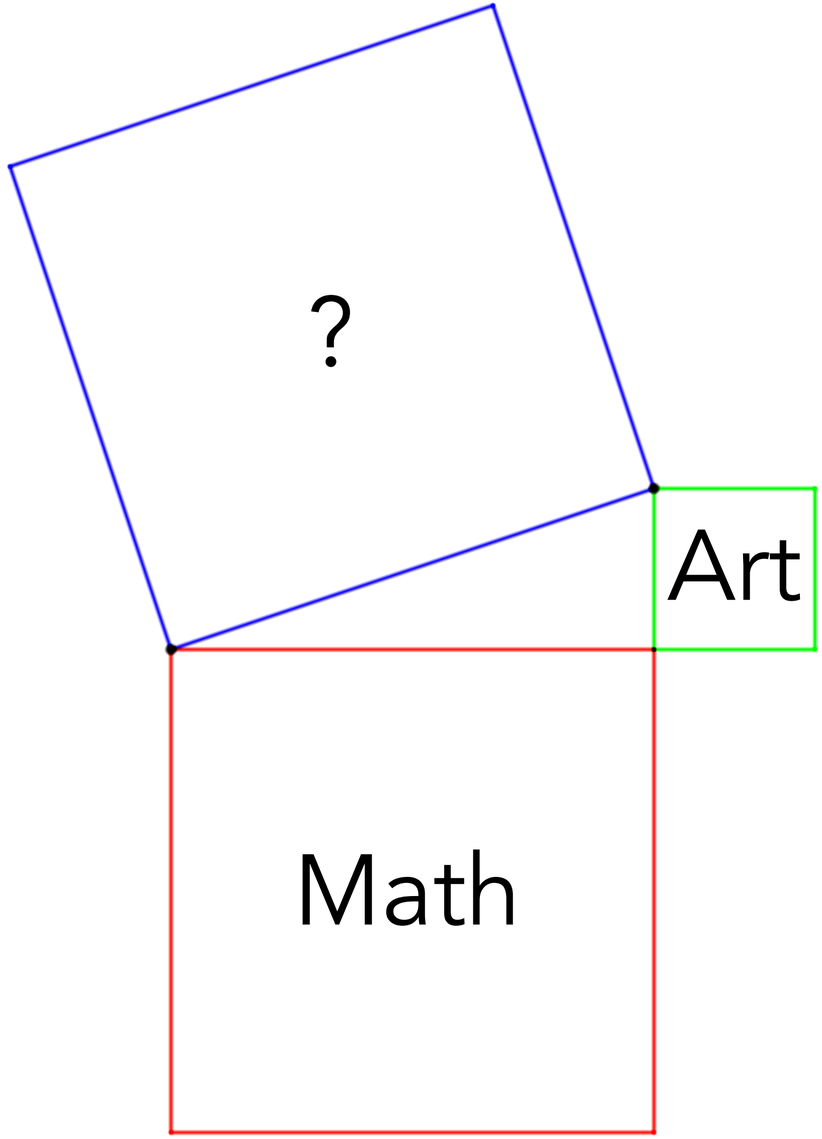
But it’s hard to measure math against art.



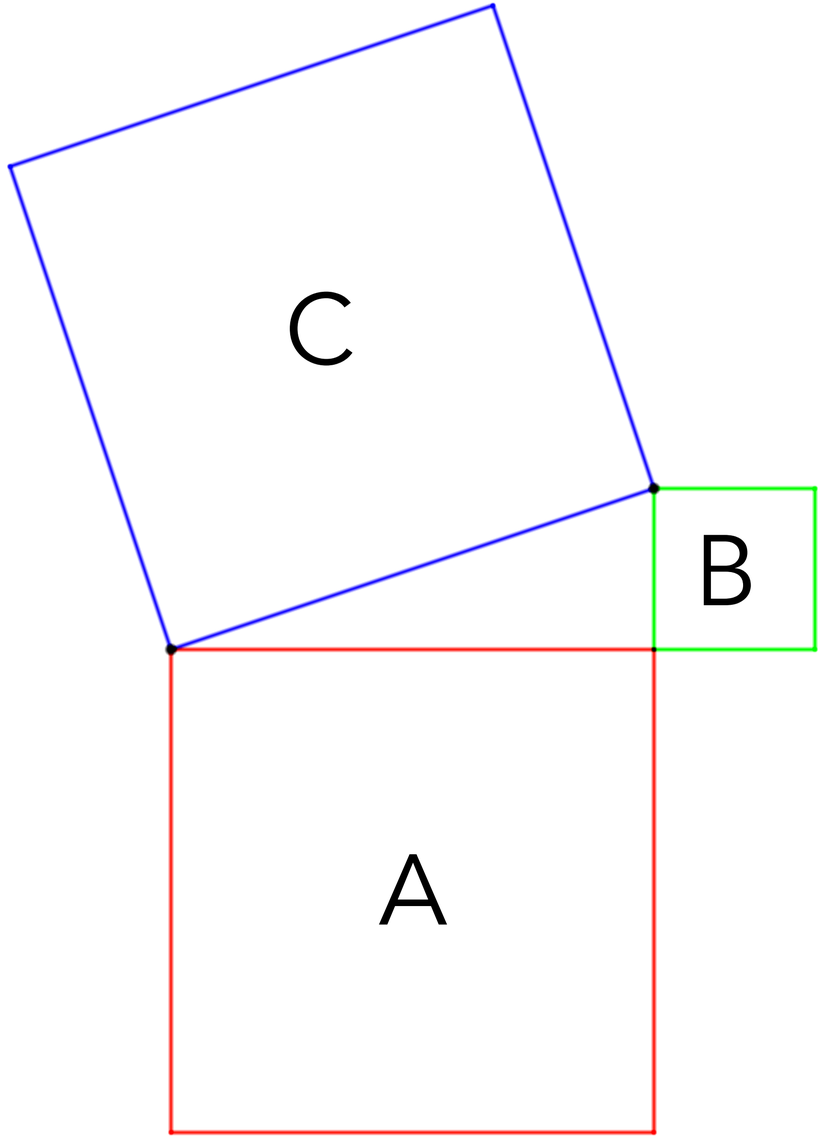
That blue line isn’t one or the other.


Math

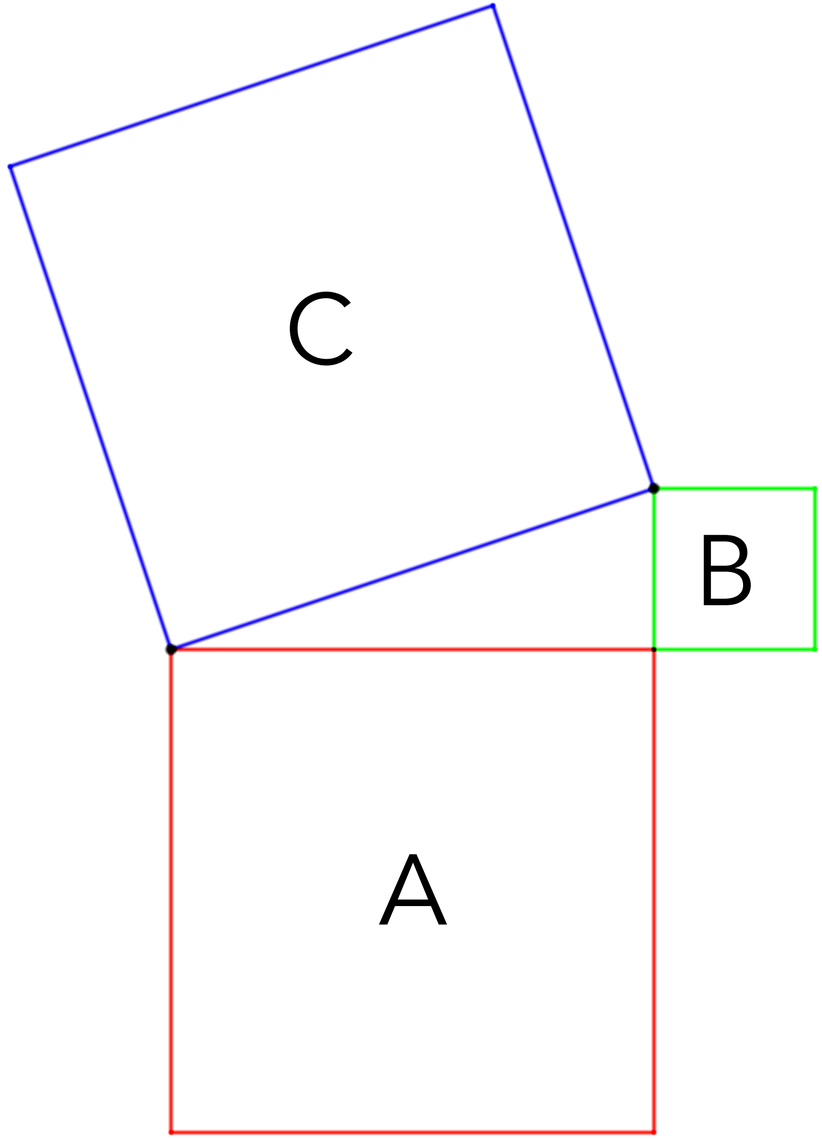

Art
Math


?
Art
Math

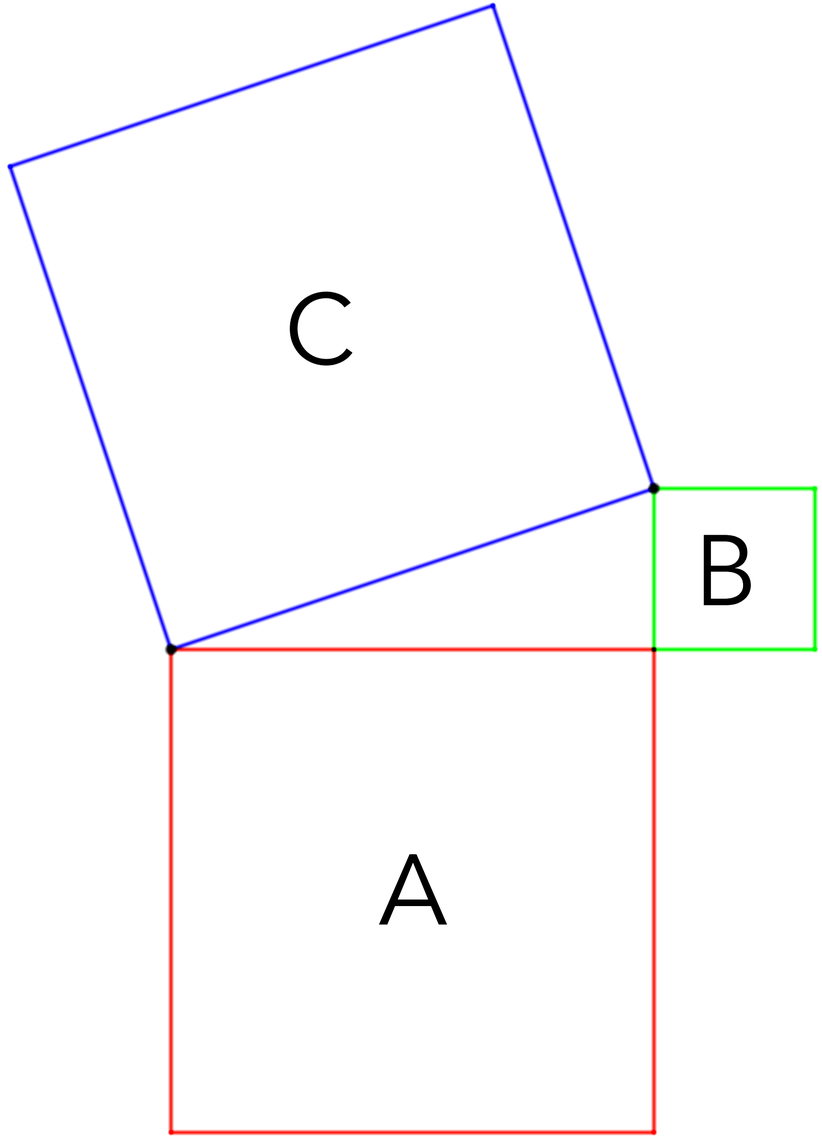
$$A*A + B*B = C*C$$

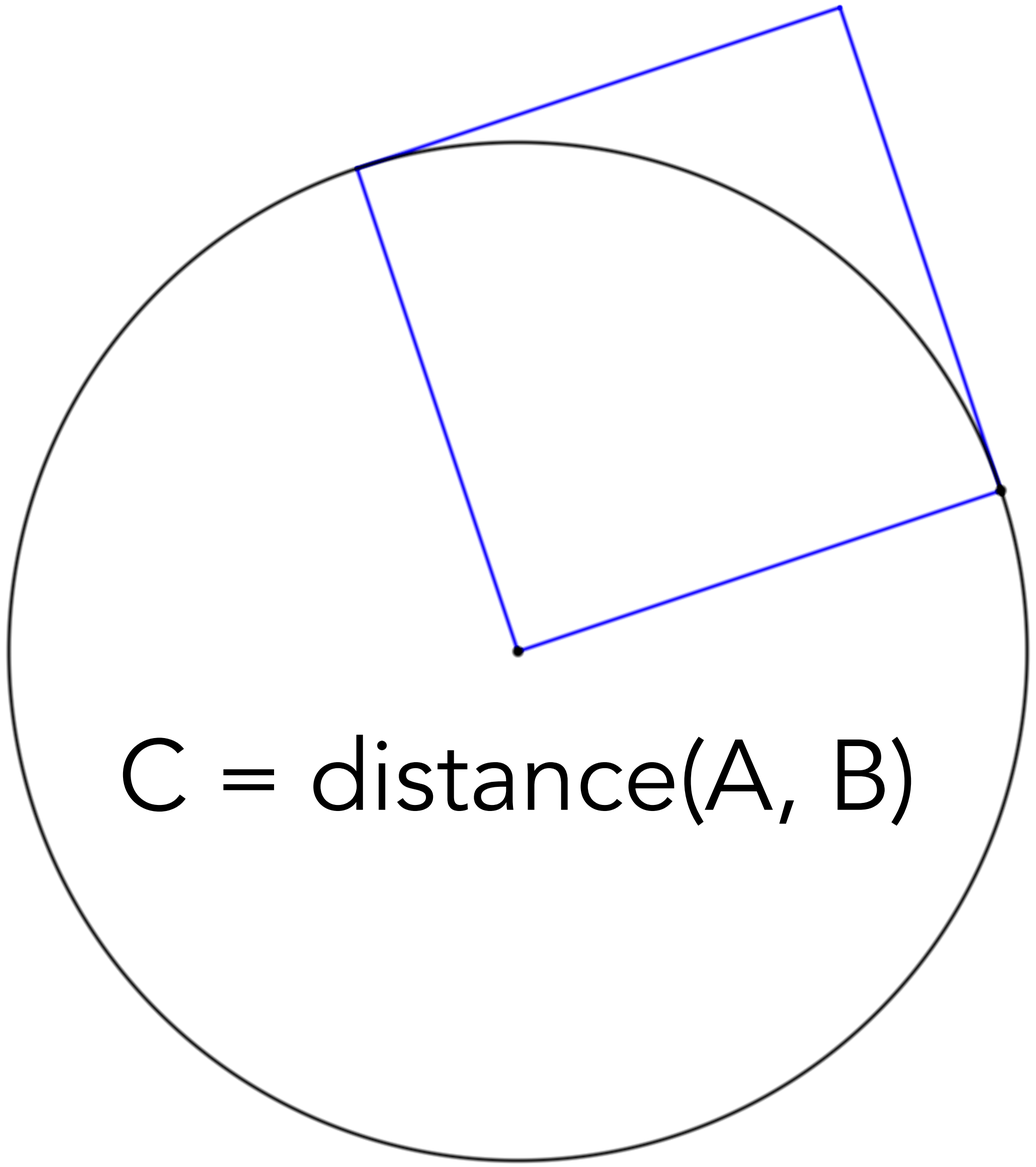


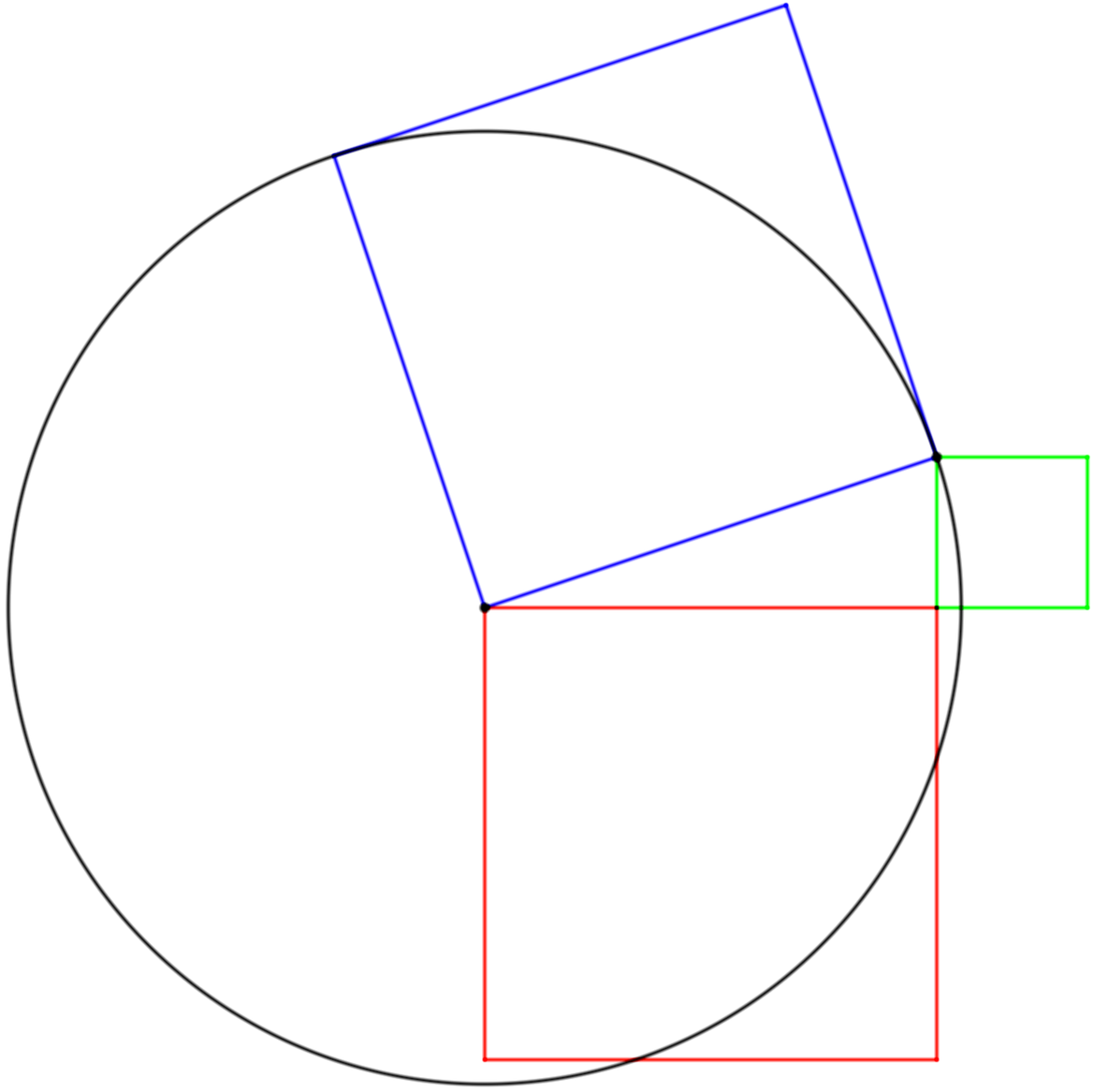
sqrt(A*A + B*B) = C



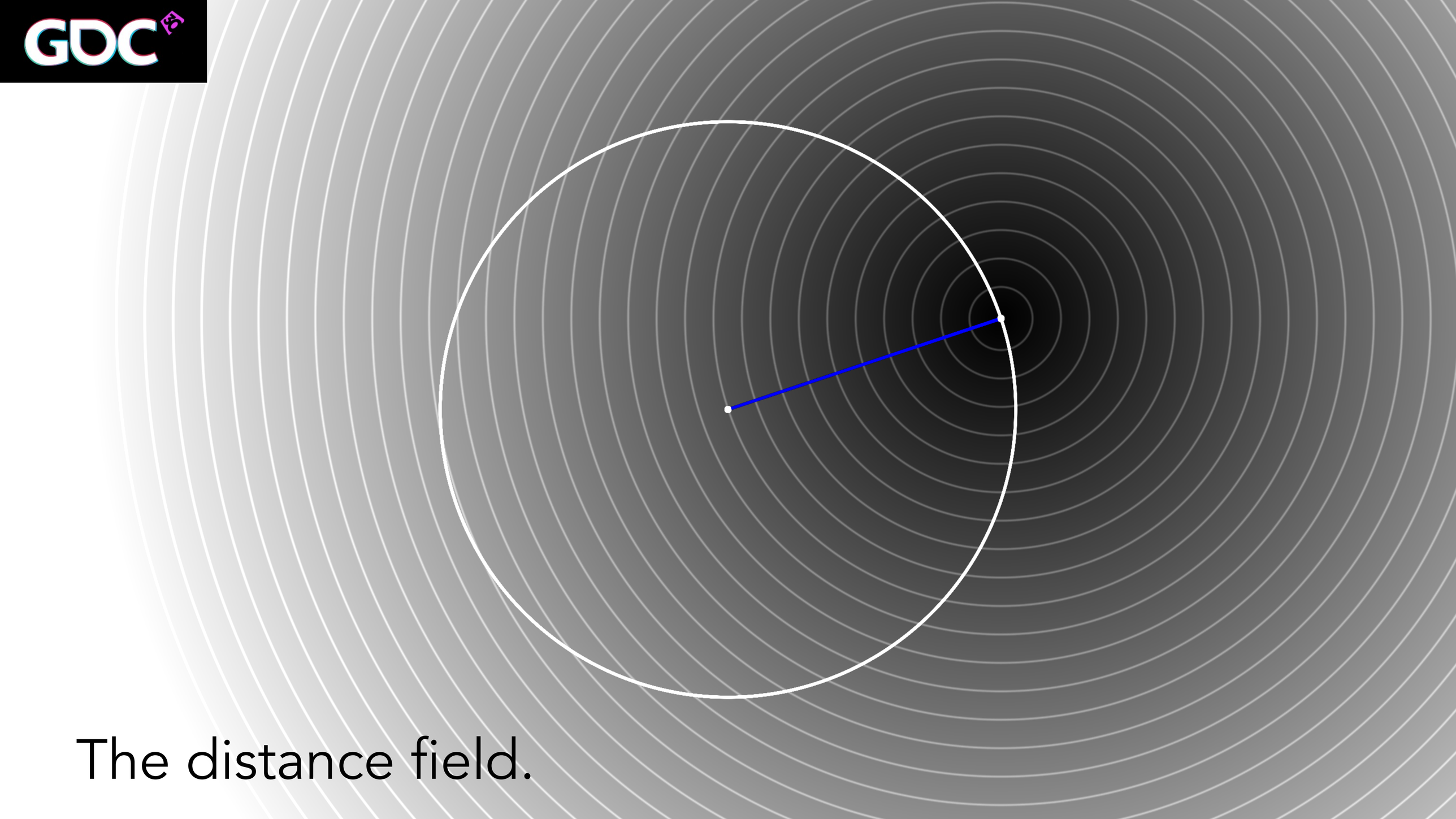
C = distance(A, B)

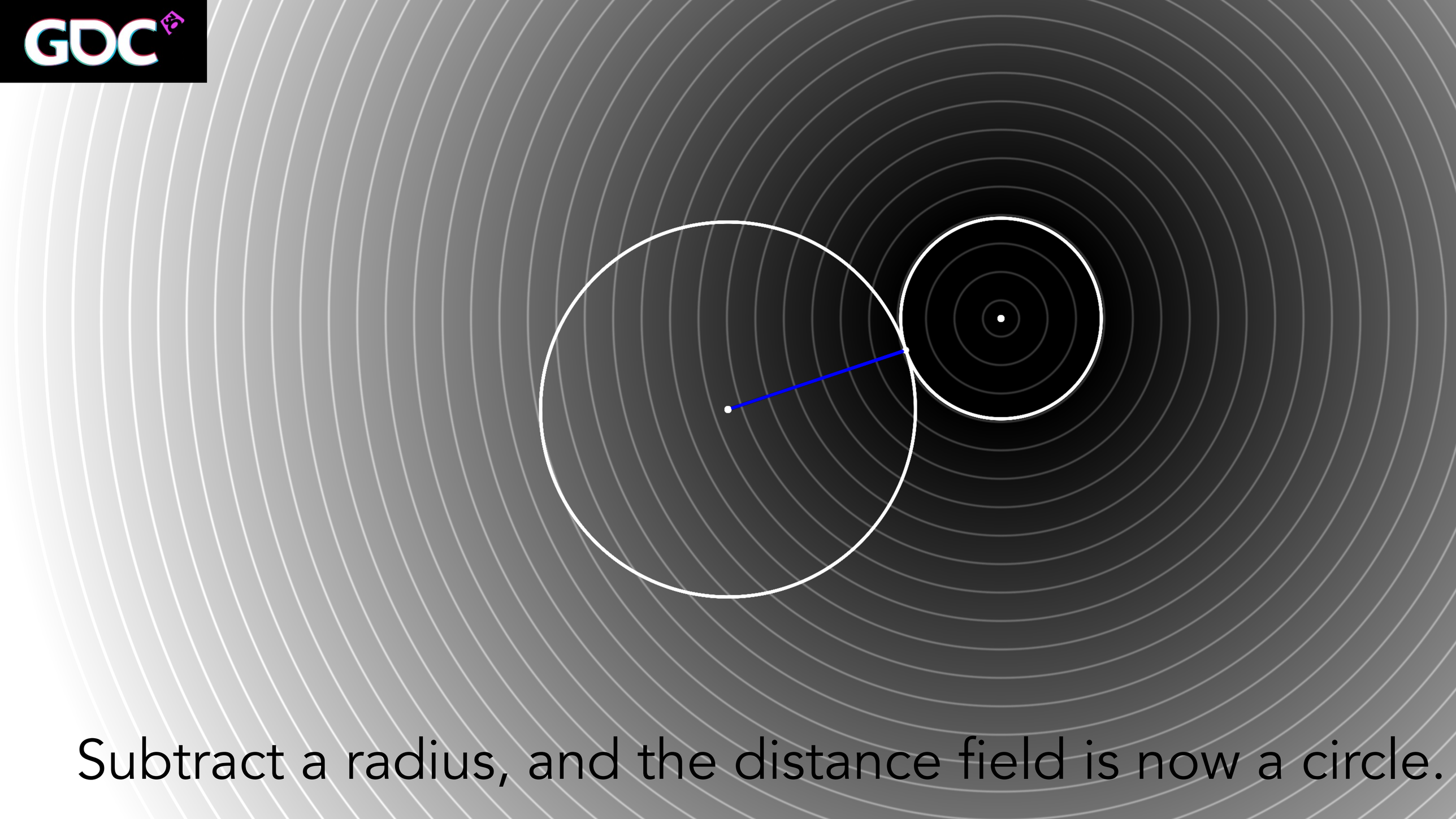



C = distance(A, B)

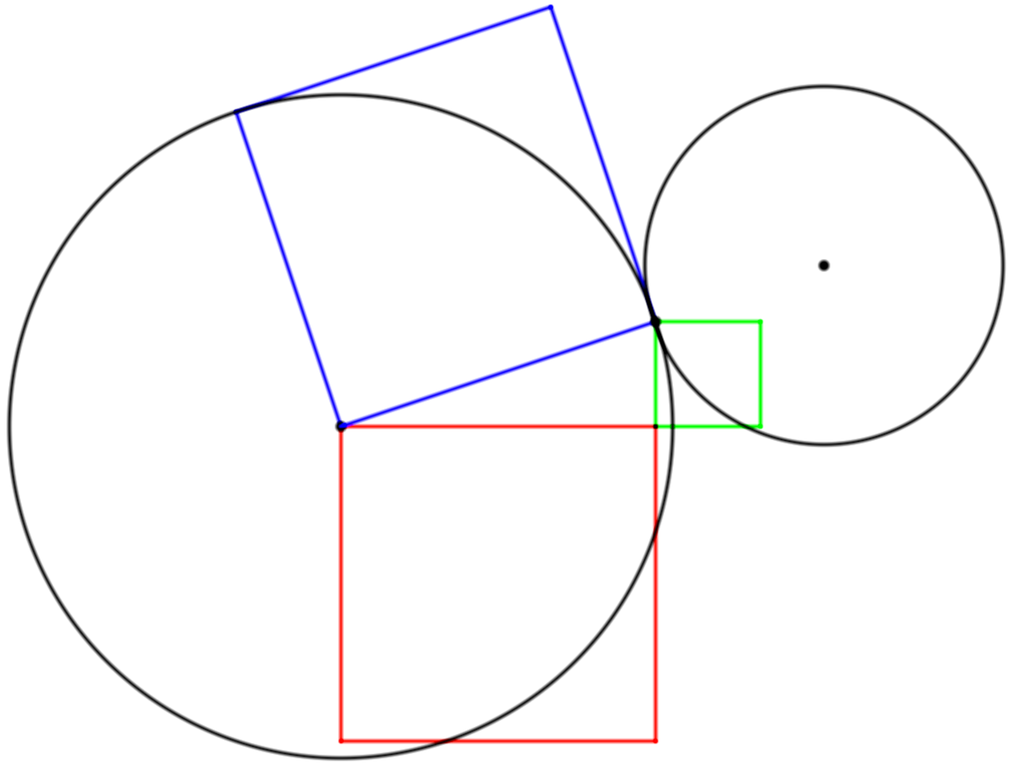


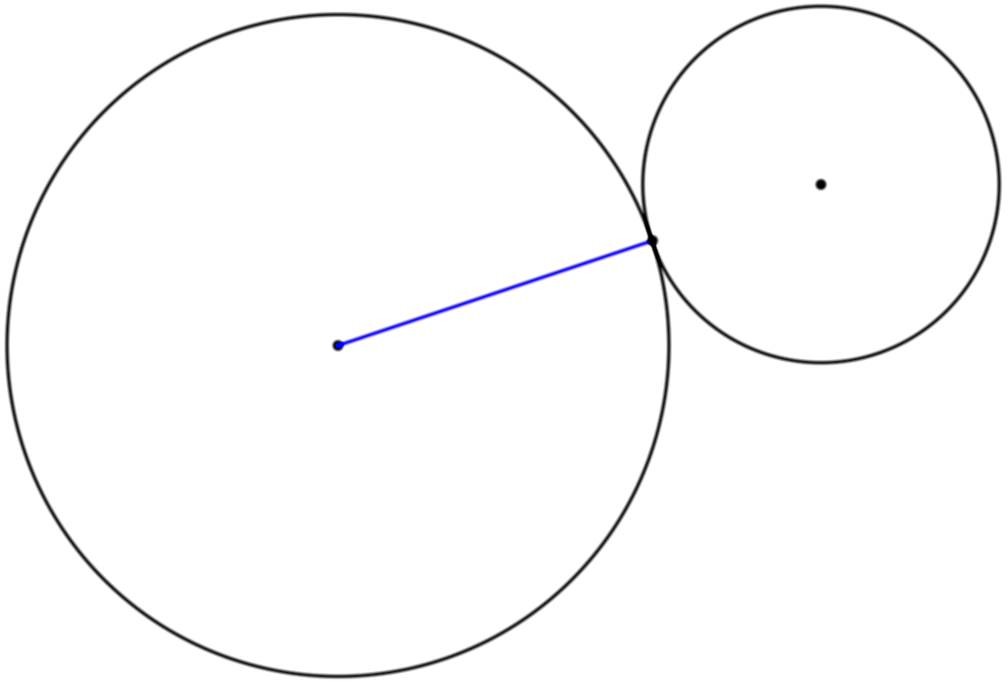
This circle is the same distance in every direction.

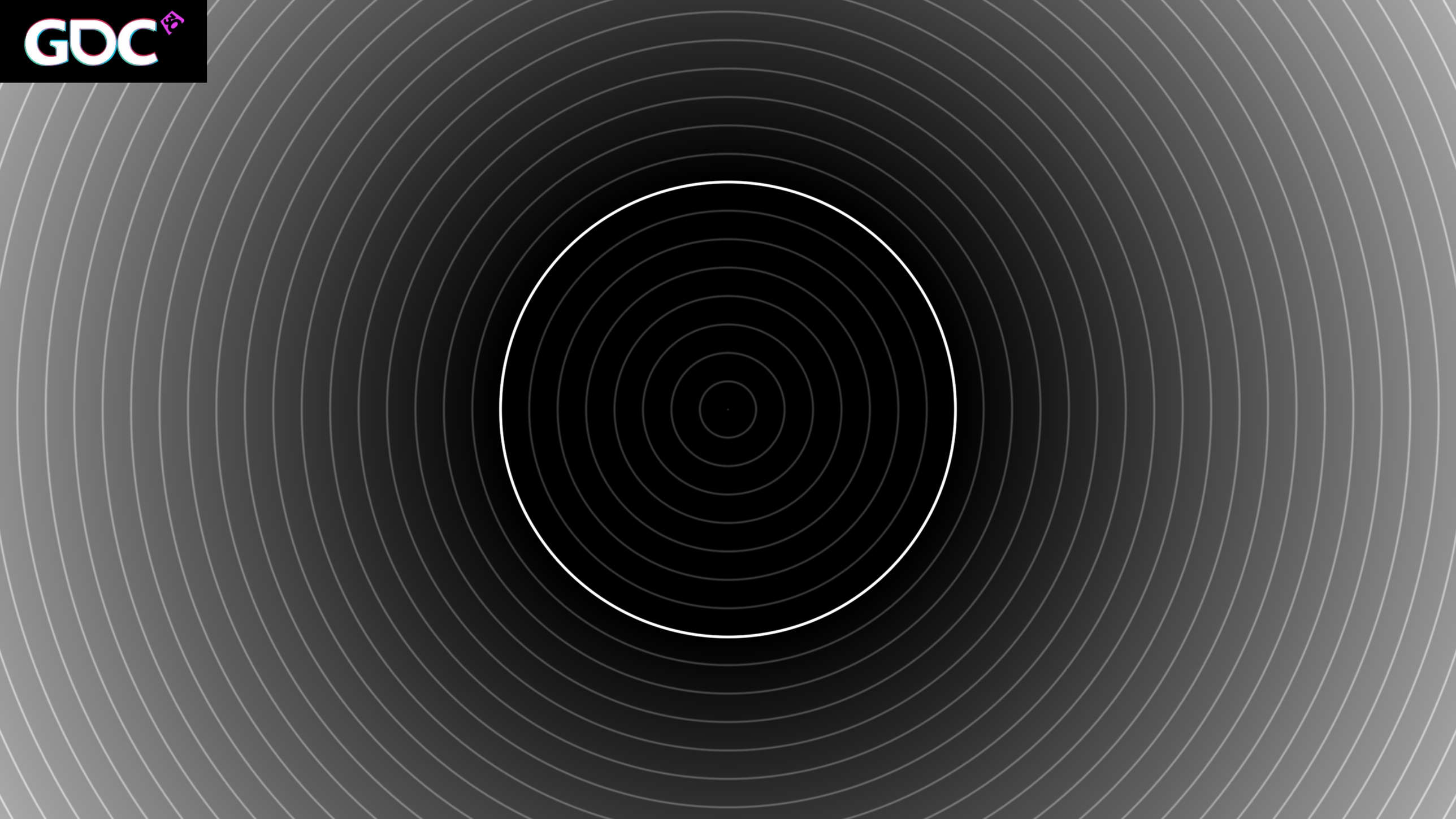

The distance field.



Subtract a radius, and the distance field is now a circle.



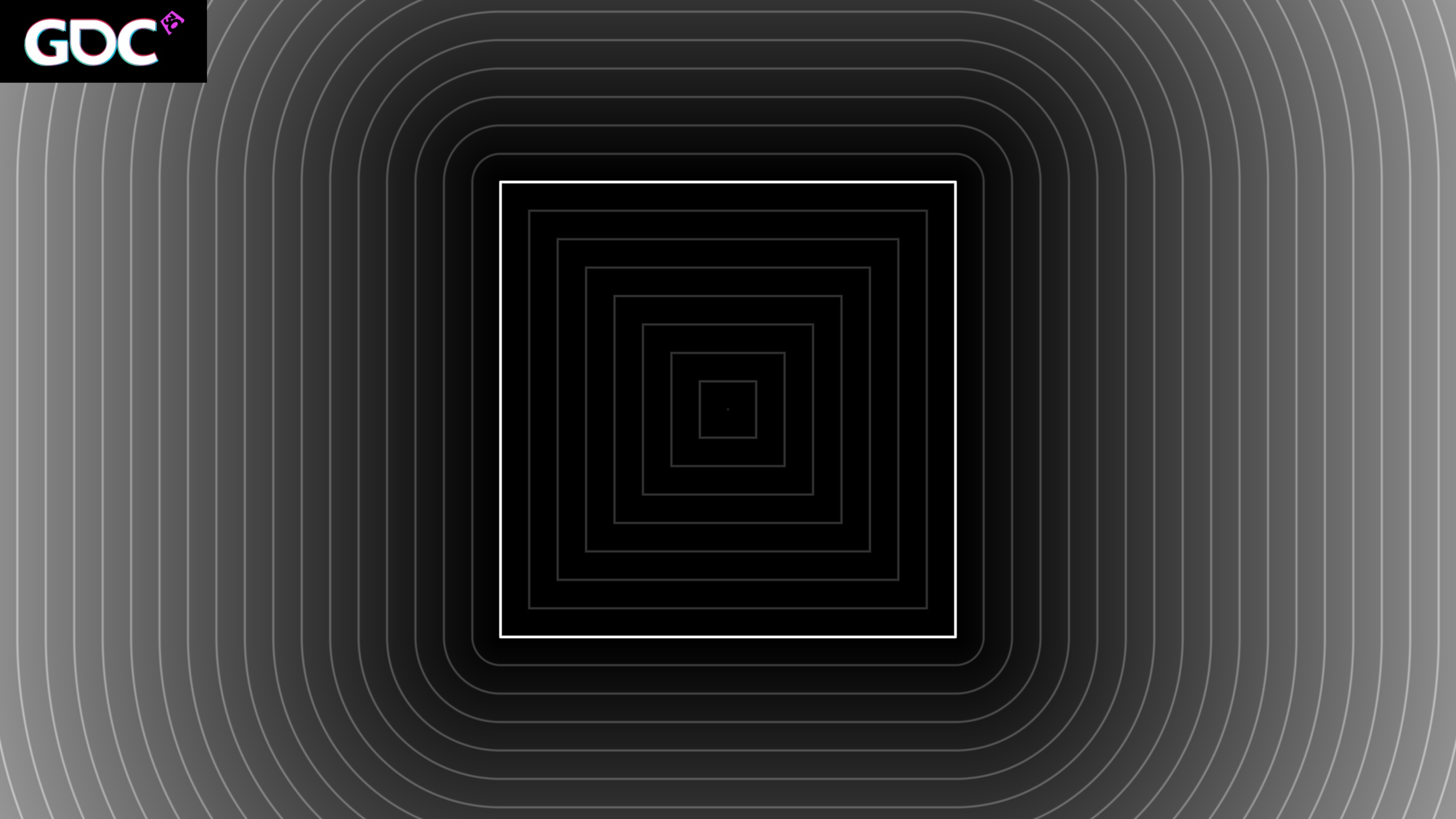




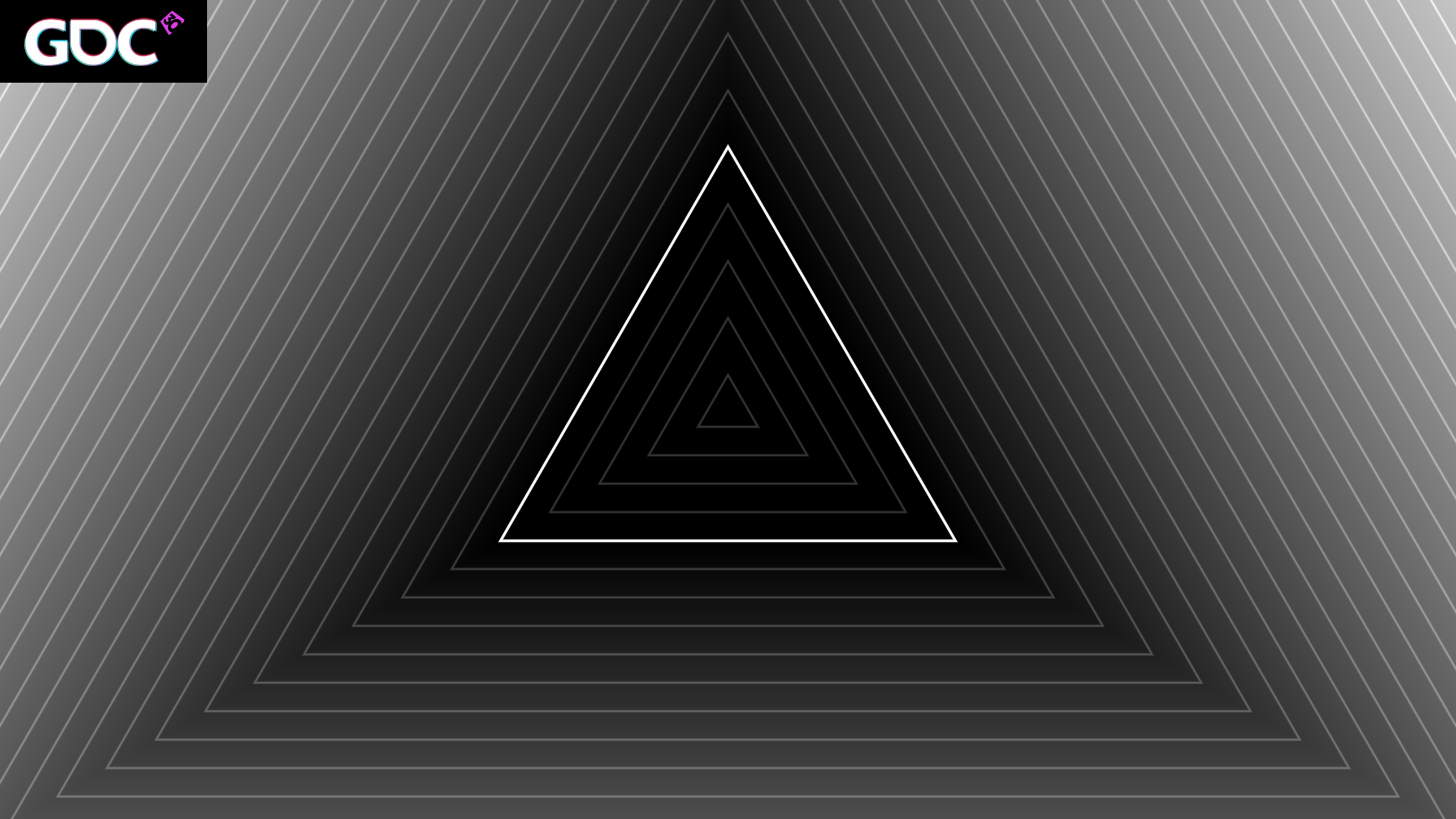
```
float sphere(vec2 position, float radius)
{
        return length(position)-radius;
}
```

```
float y(float height)
{
        return position.y - height;
}
```

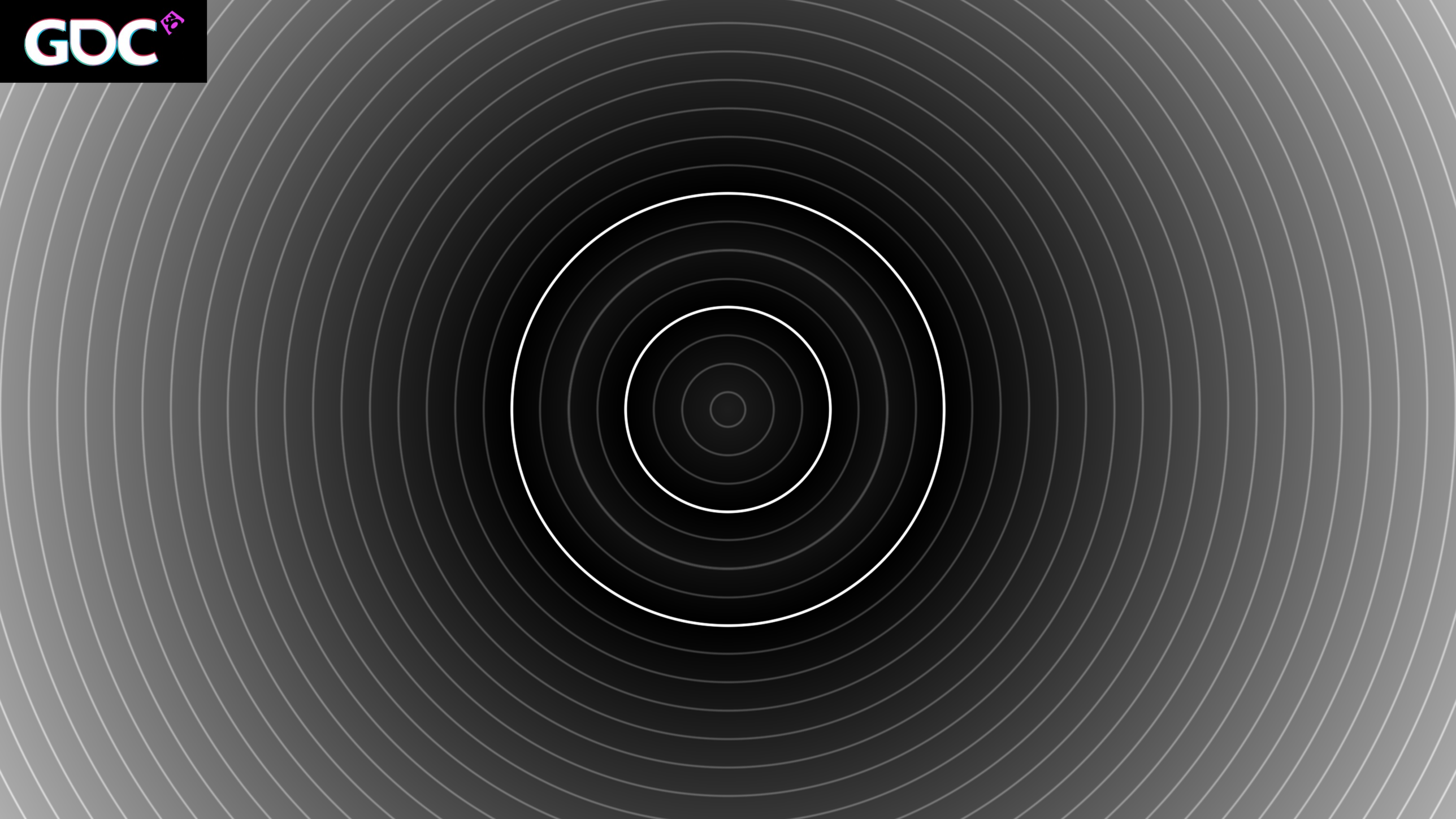
```
float x(float offset)
{
        return position.y - offset;
}
```



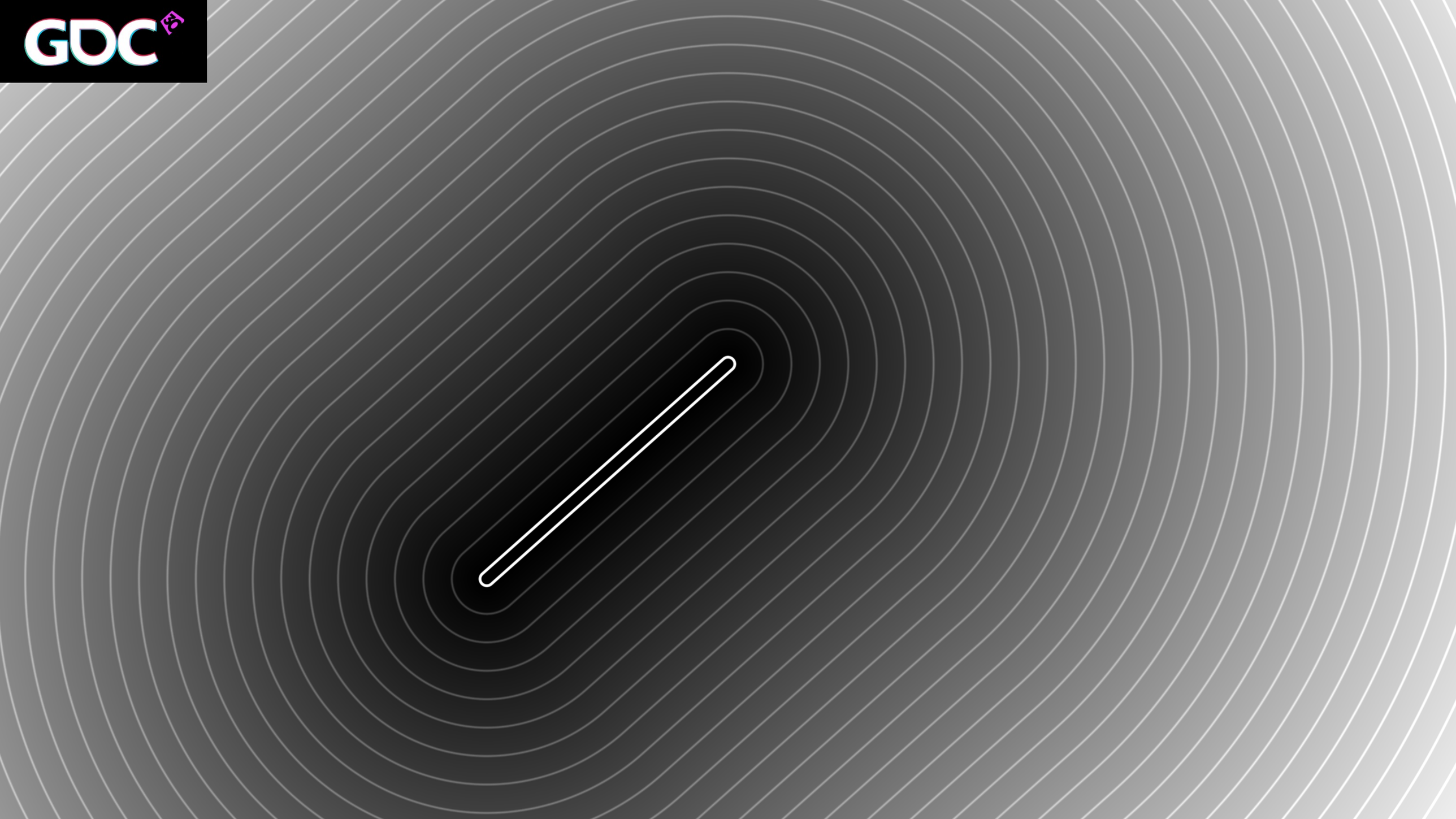
```
float cube(vec2 position, vec2 scale)
{
        vec2 vertex     = abs(position) - scale;
        vec2 edge       = max(vertex, 0.);
        float interior  = max(vertex.x, vertex.y);

        return min(interior, 0.) + length(edge);
}
```



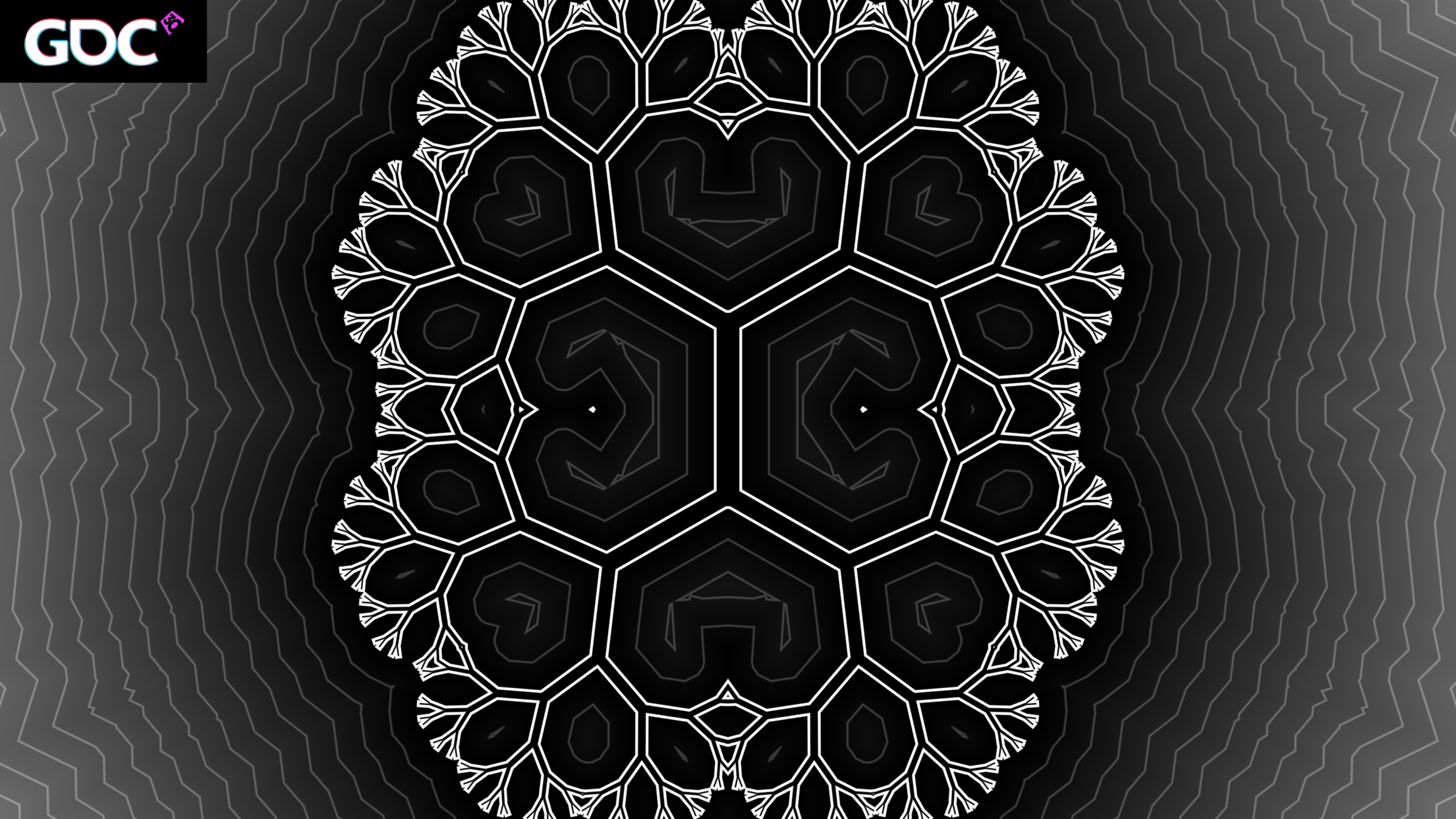
```
float prism(vec2 position, float scale)
{
        position.y    *= 0.57735026918; //1./sqrt(3.);
        scale         *= 0.57735026918;

        vec3 edge      = vec3(0.);
        edge.x         = position.y + position.x;
        edge.y         = position.x - position.y;
        edge.z         = position.y + position.y;
        edge          *= .86602540358;  //sqrt(3.)/2.;


        return max(edge.x, max(-edge.y, -edge.z))-scale;
}
```



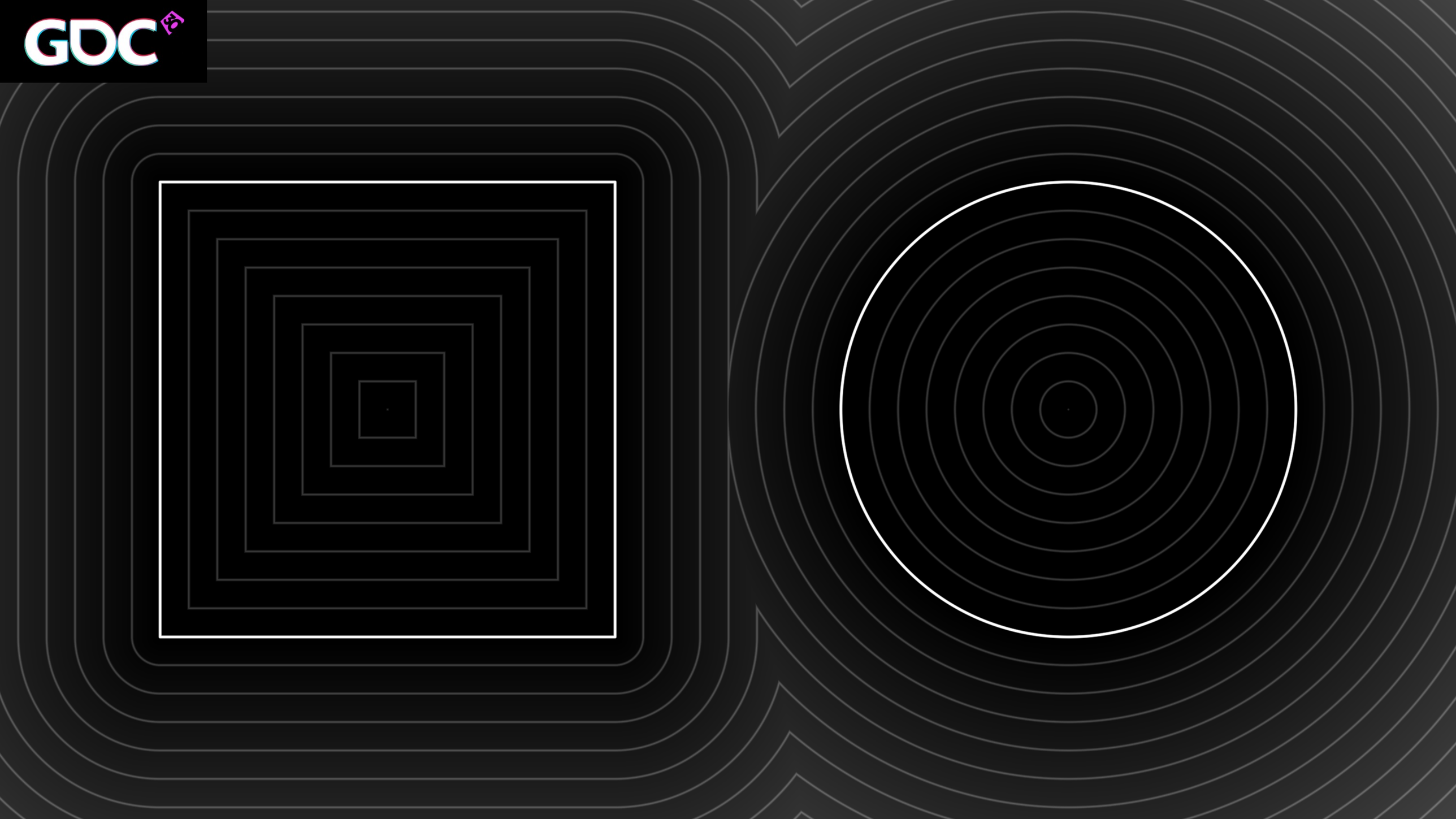
```
float torus(vec2 position, vec2 radius)
{
        return abs(abs(length(position)-radius.x)-radius.y);
}
```

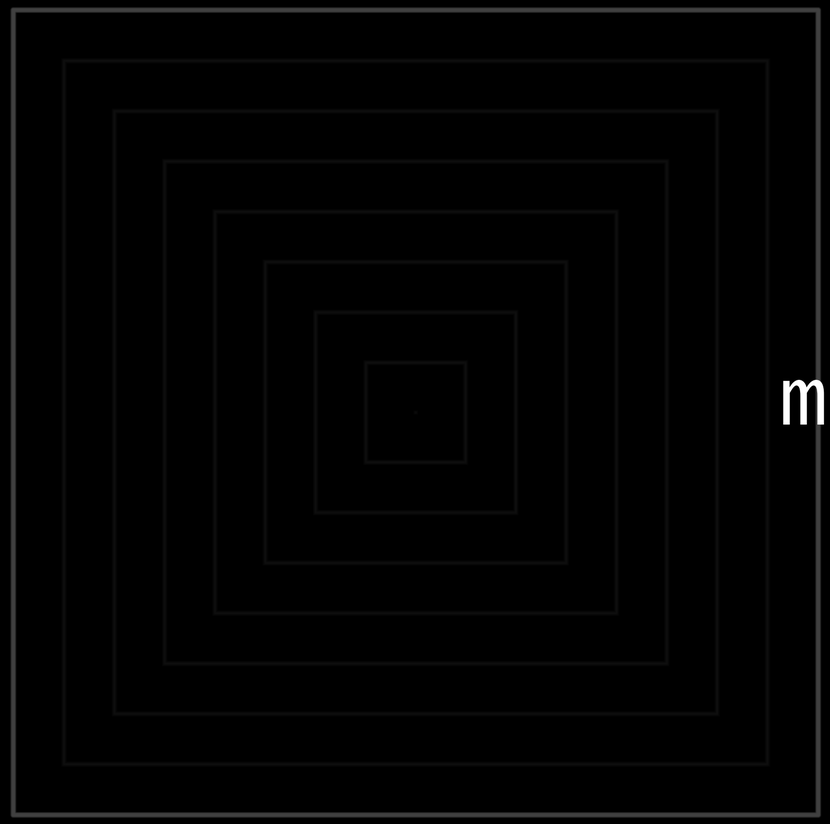


```
float torus(vec2 position, vec2 radius)
{
        return abs(abs(length(position)-radius.x)-radius.y);
}
```

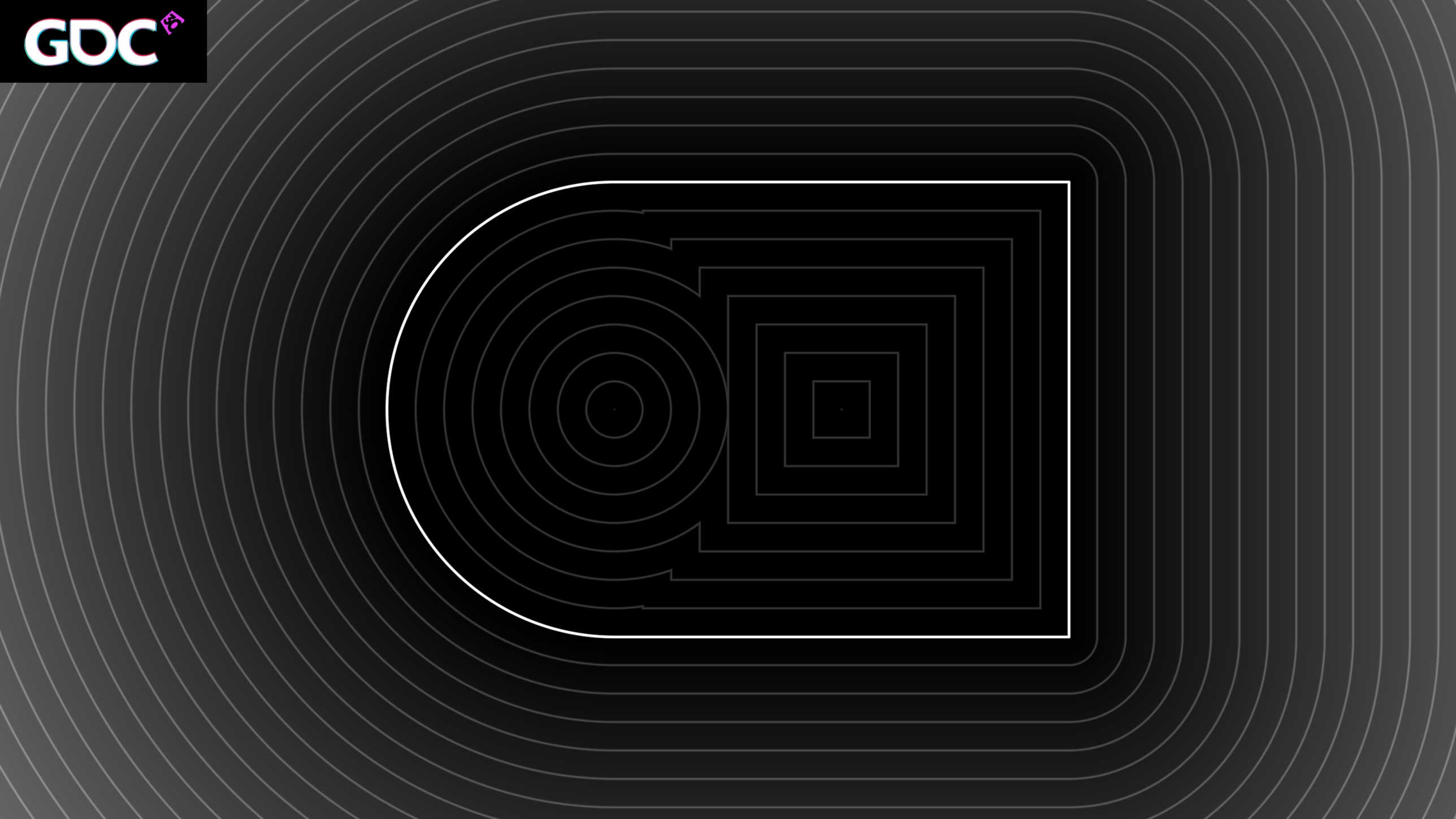

GDC

```
float fractal(vec2 position, float rotation, float translation, float scale)
{
	const int iterations	= 8;

	float radius		= 1.5/float(iterations);

	rotation		= rotation * TAU + TAU * .5;

	position		= abs(position);
	position.y		-= radius;
	scale			*= radius;
	float result		= max(position.x, position.y) - scale;

	for (int i = 0; i < iterations; i++)
	{
		float magnitude	= length(position);

		float phase	= atan(position.x, position.y);
		phase		= mod(phase, rotation) - rotation * .5;

		position.x	= magnitude * cos(phase);
		position.y	= magnitude * sin(phase);

		position	= abs(position);

		position.y	-= translation;

		translation	*= .7;
		scale		*= .8;
		rotation	*= -.98;

		result		= min(max(position.x, position.y) - scale, result);
	}

	return result;
}
```

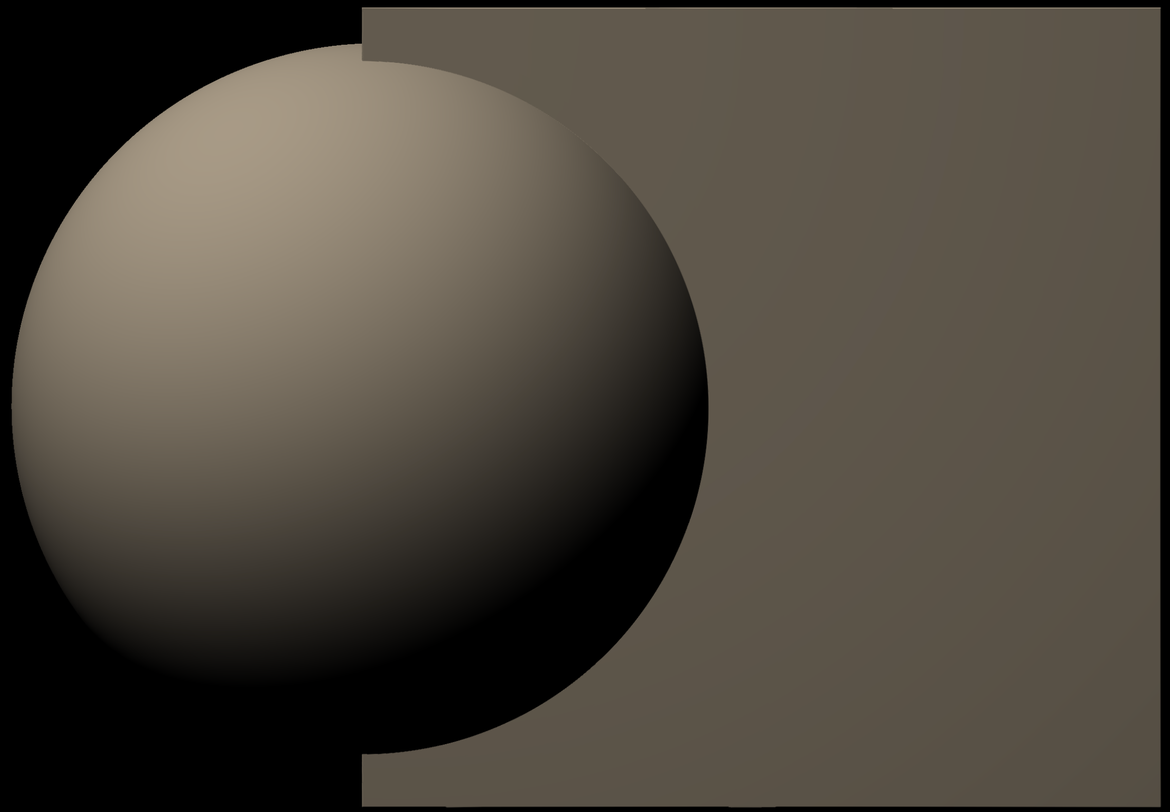


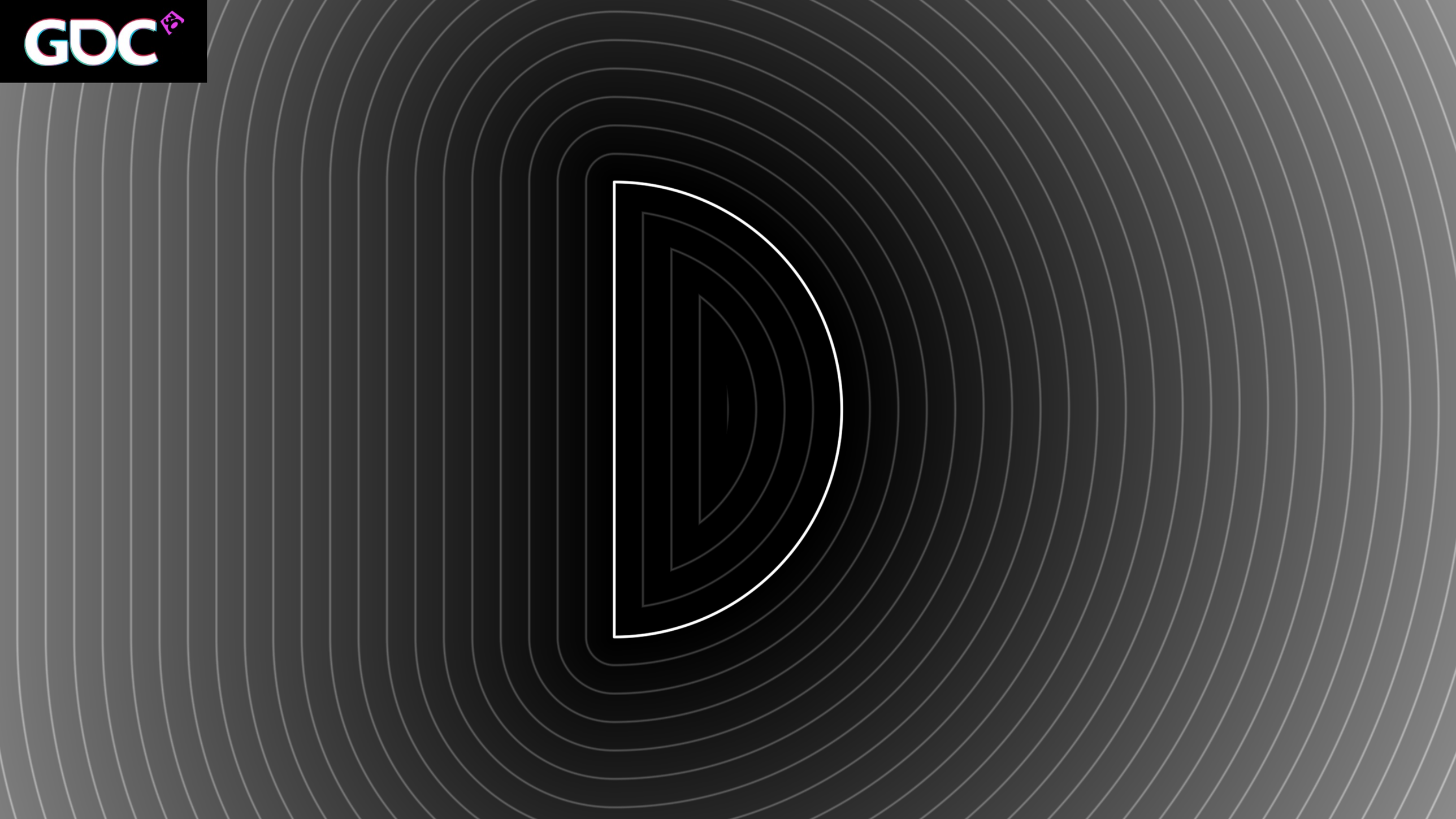


```
min(a, b)
```

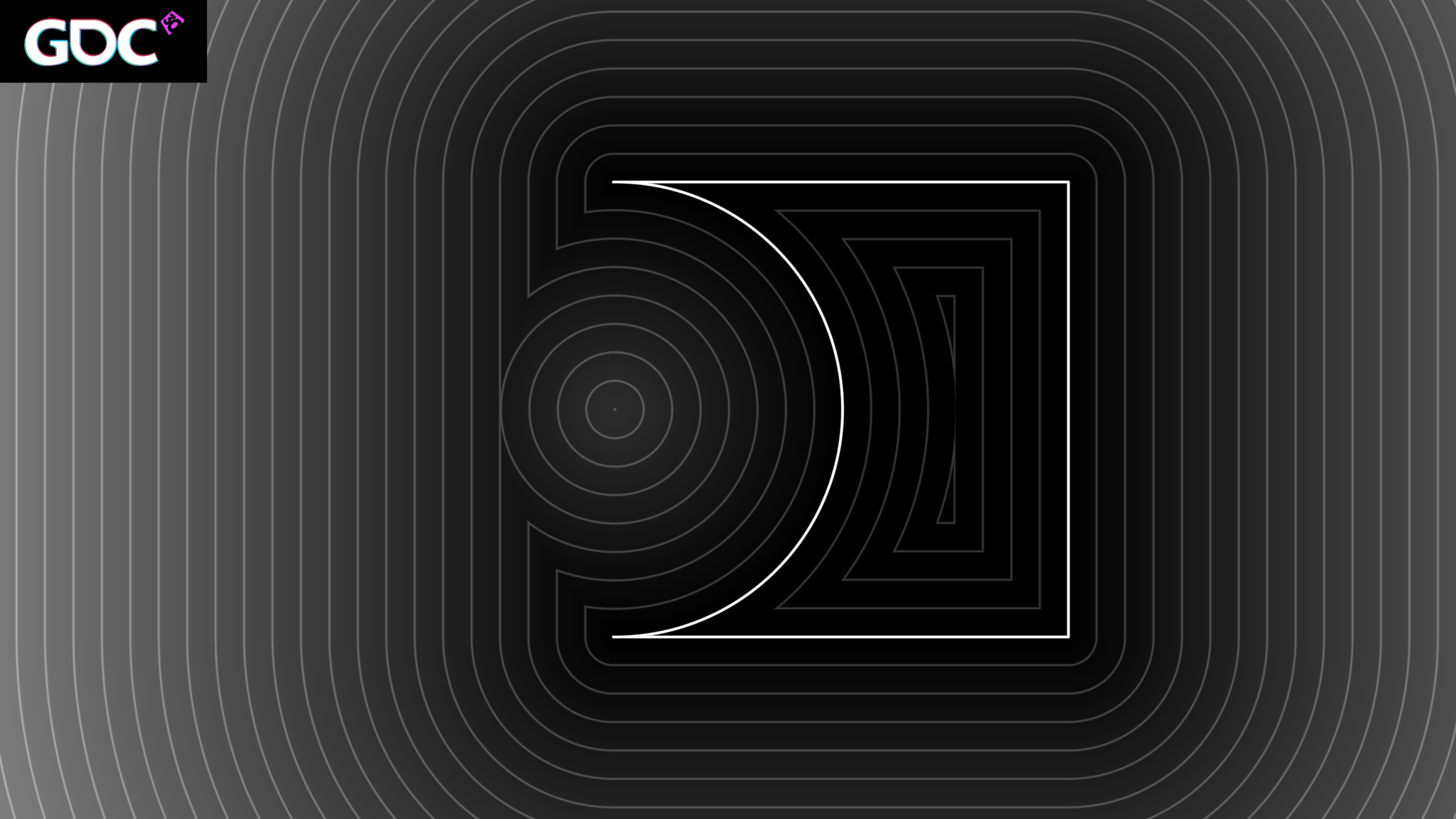


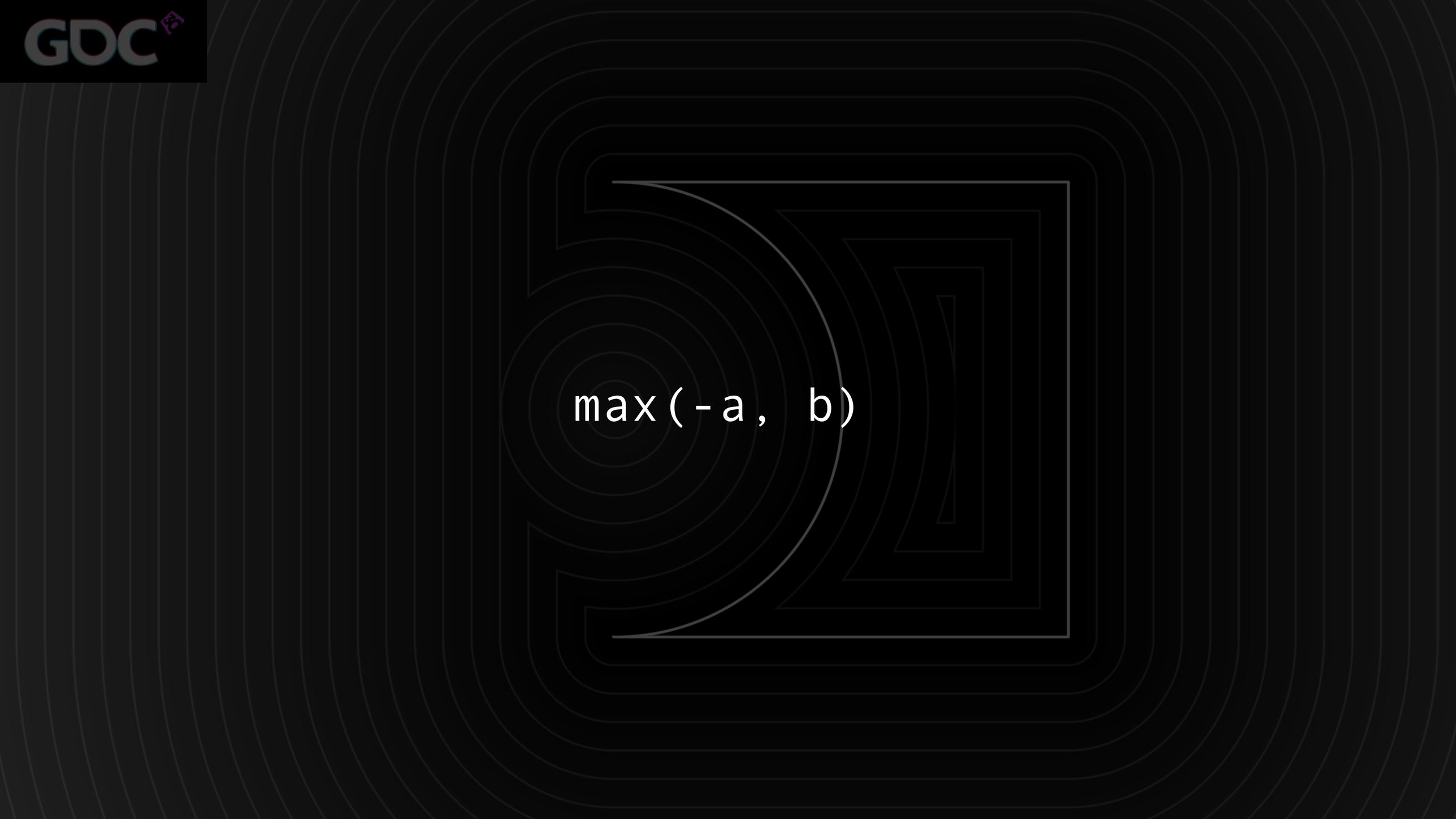
min(a, b)

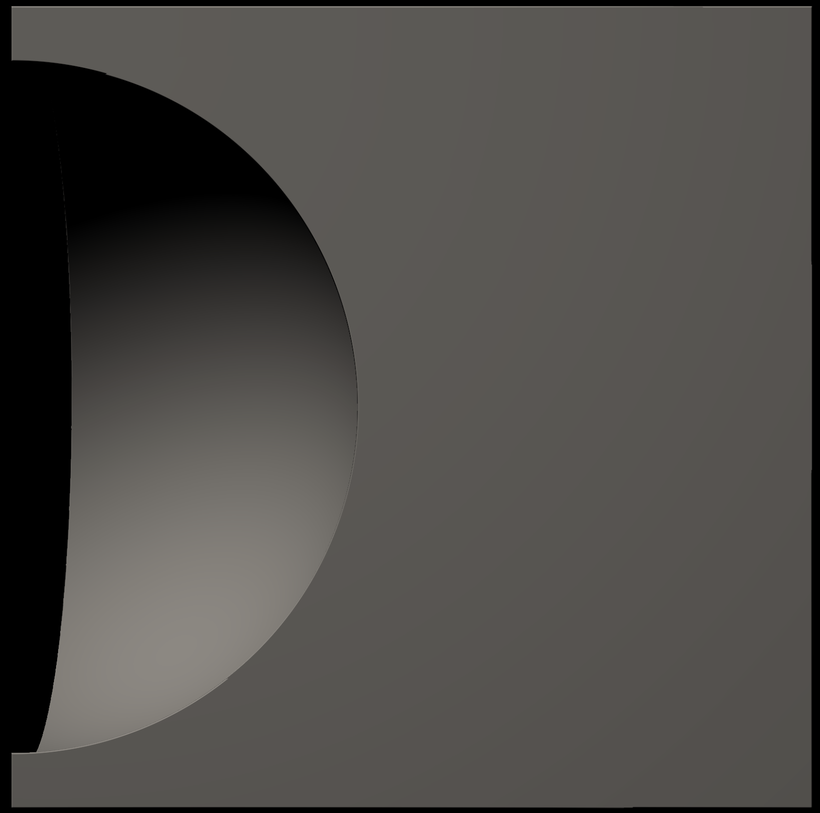




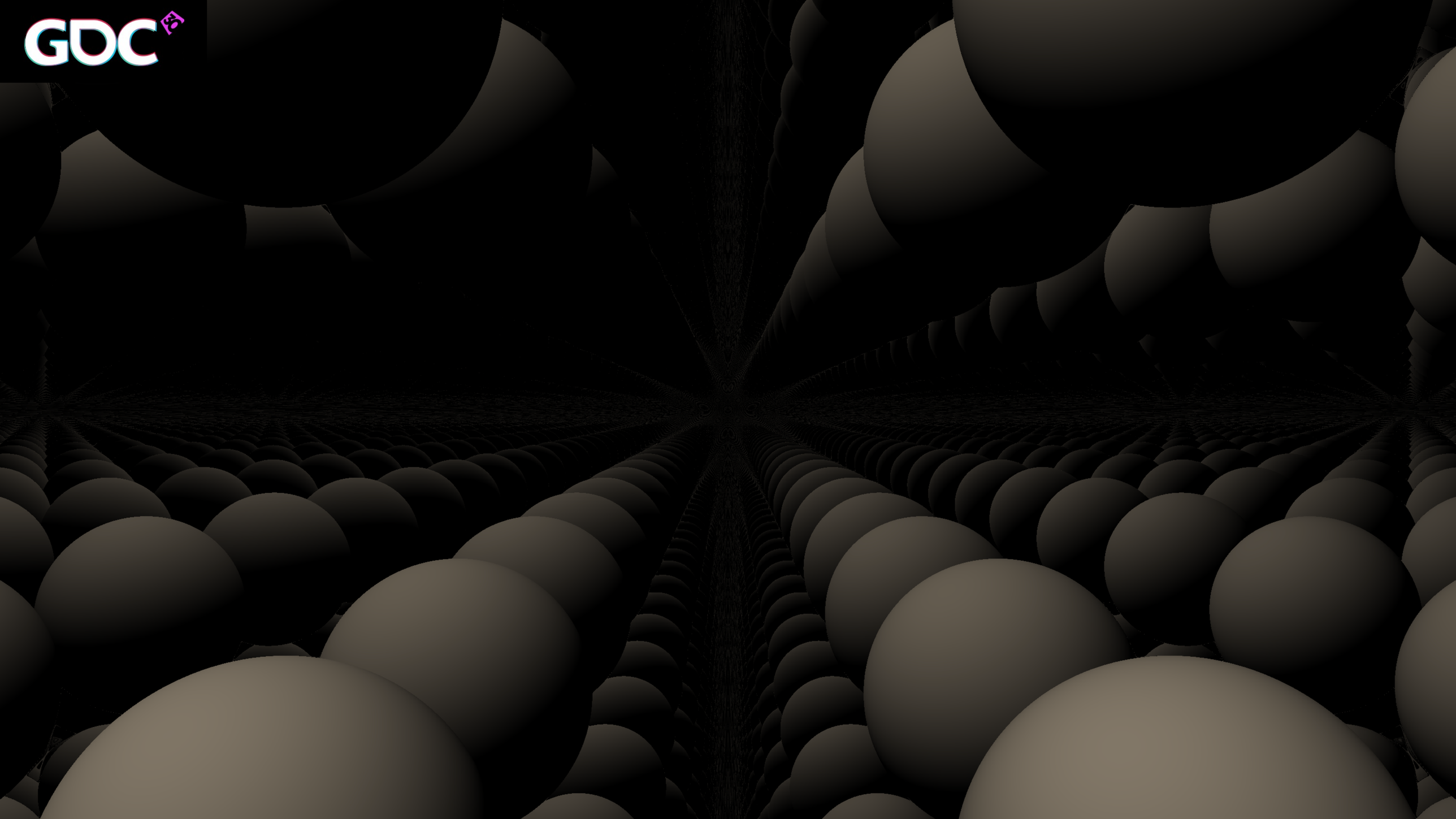
max(a, b)

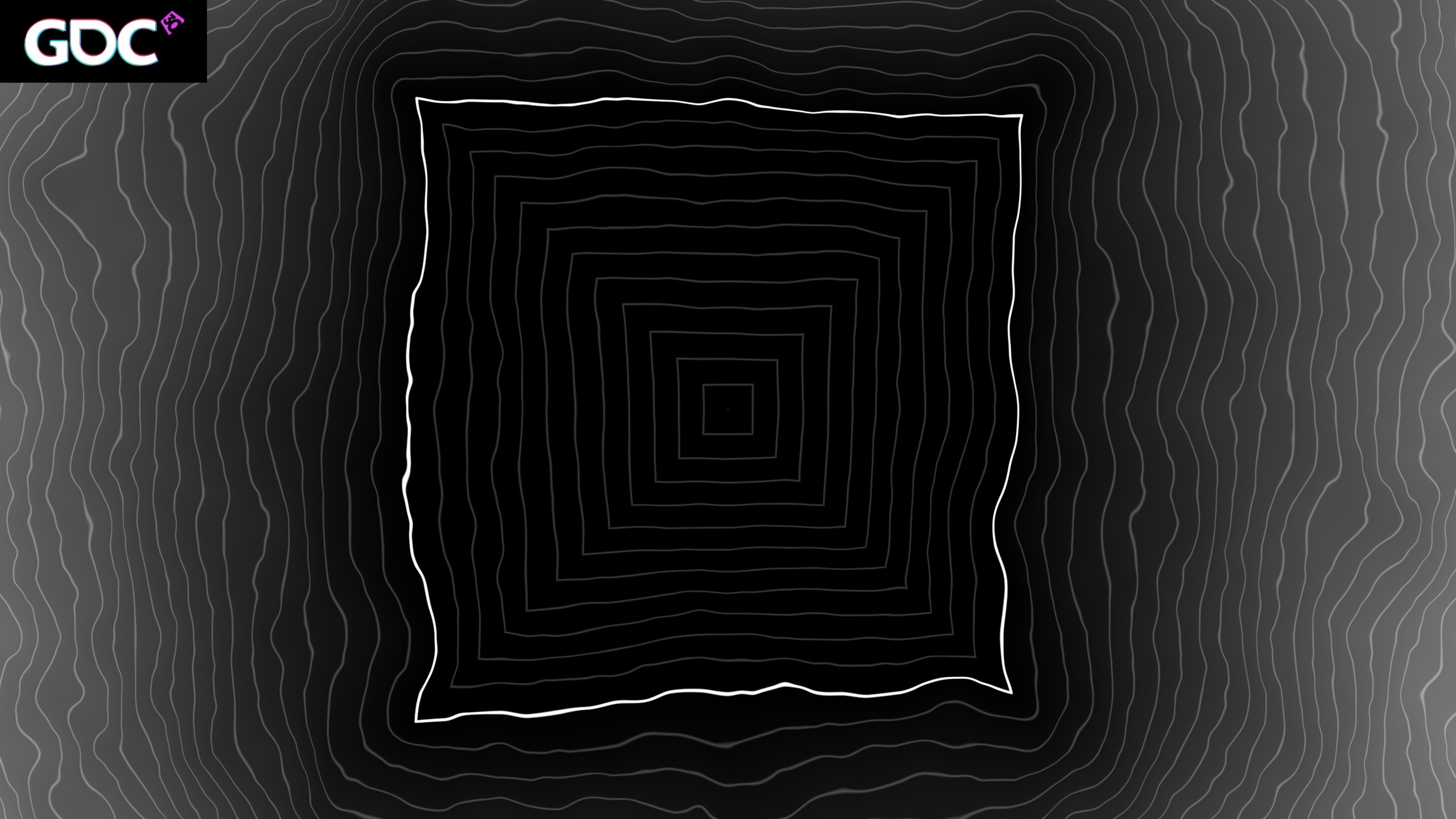



max(-a, b)

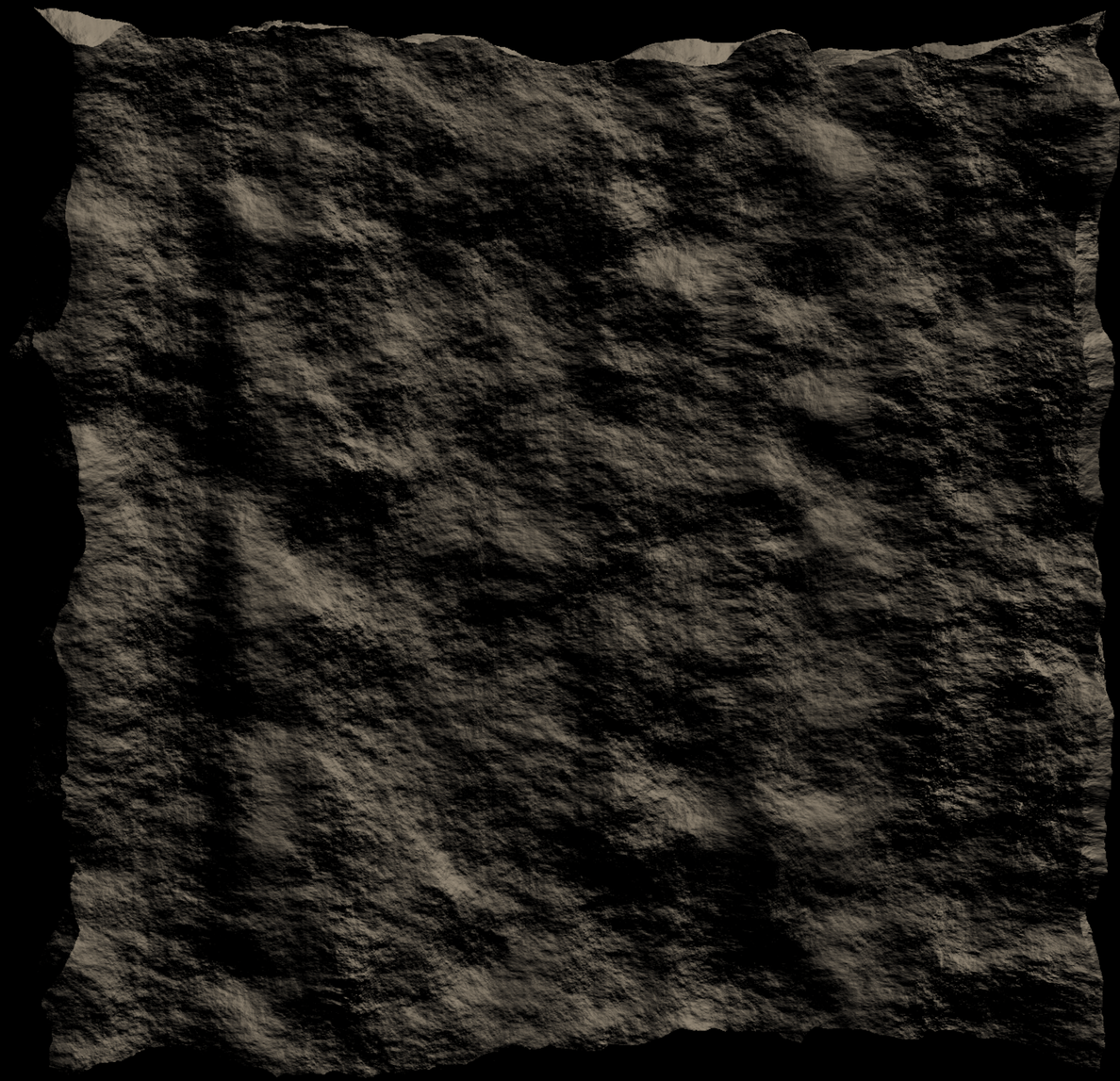


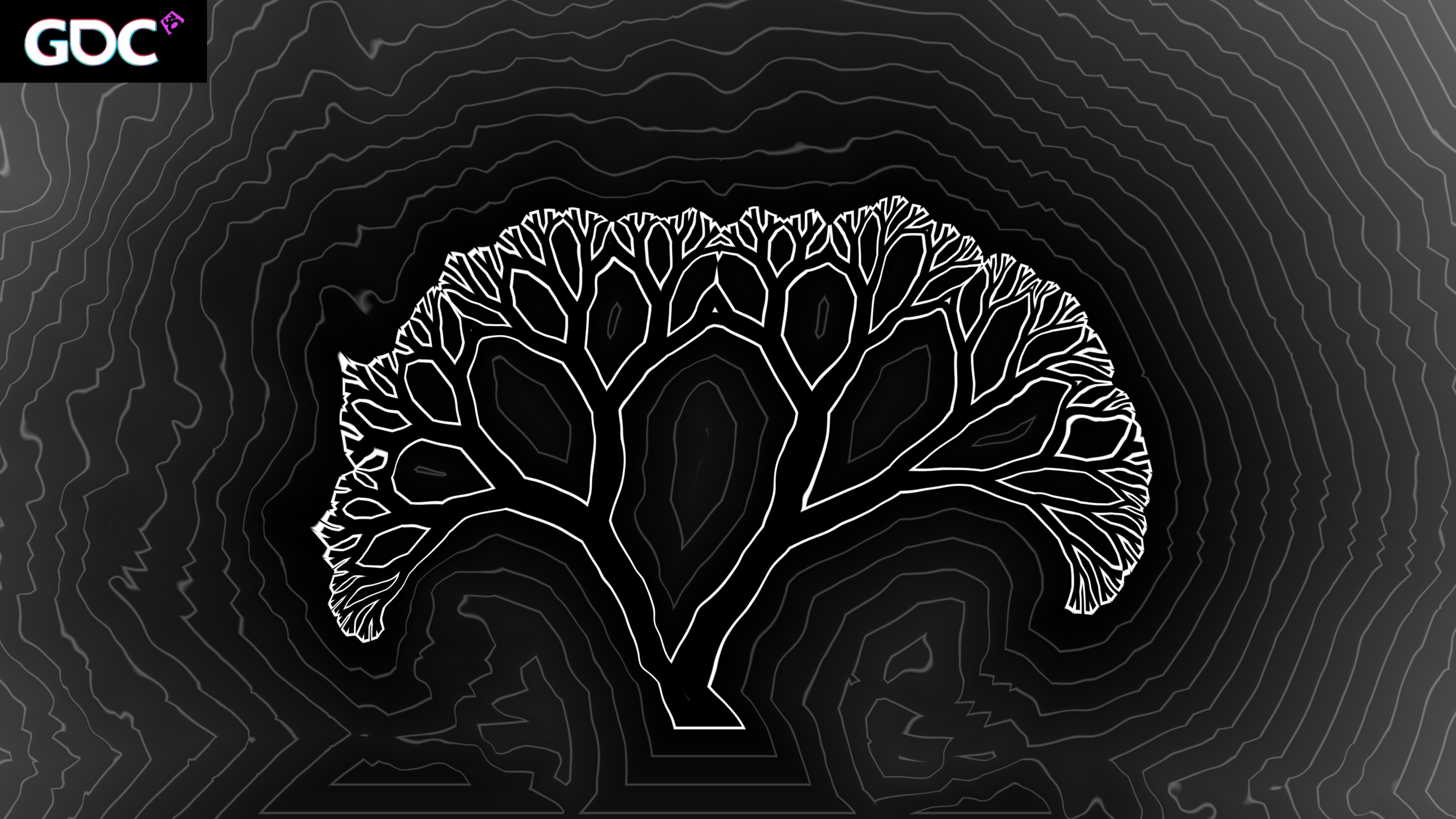
```
mod(position, scale) - scale * .5
```

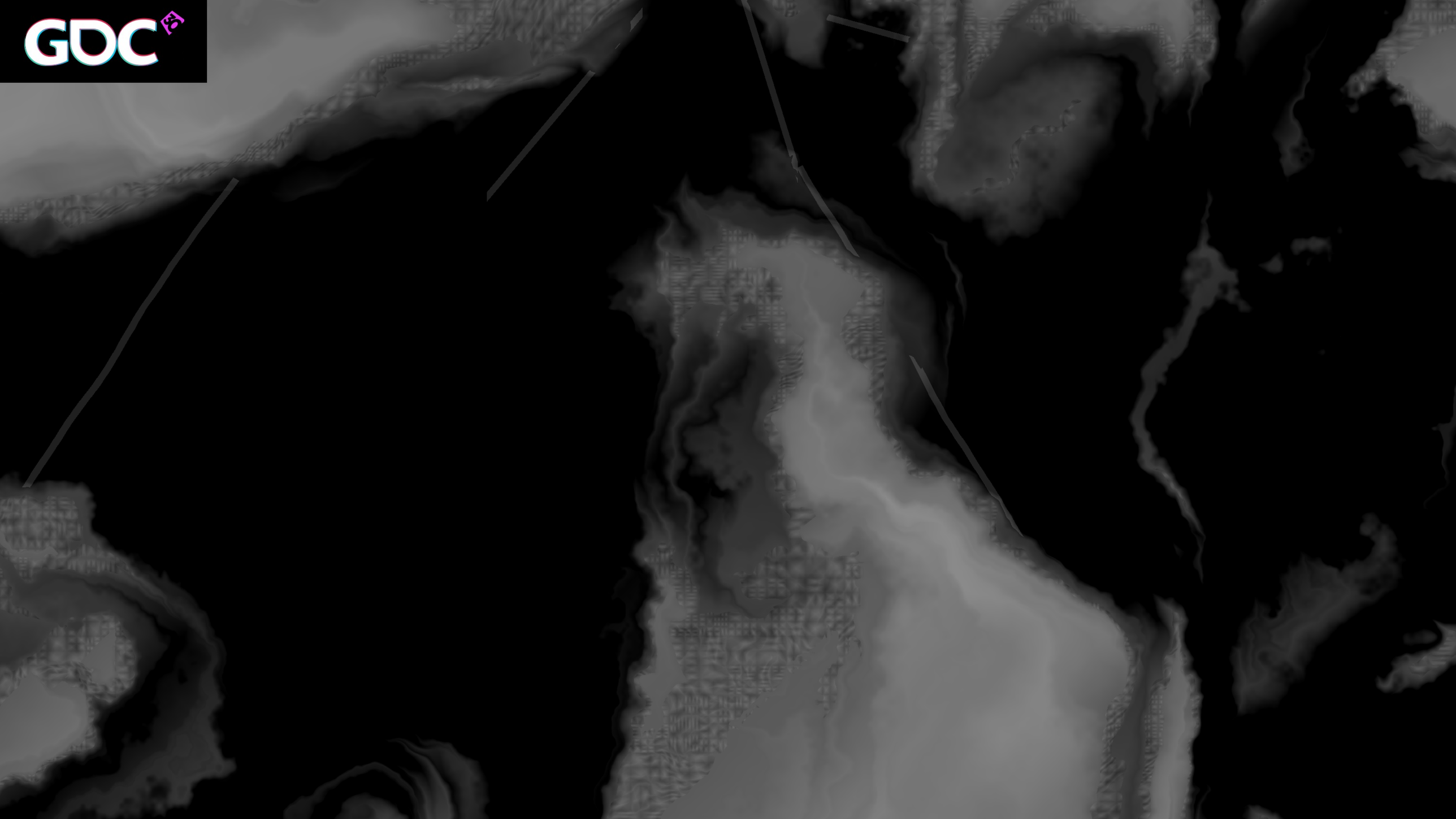


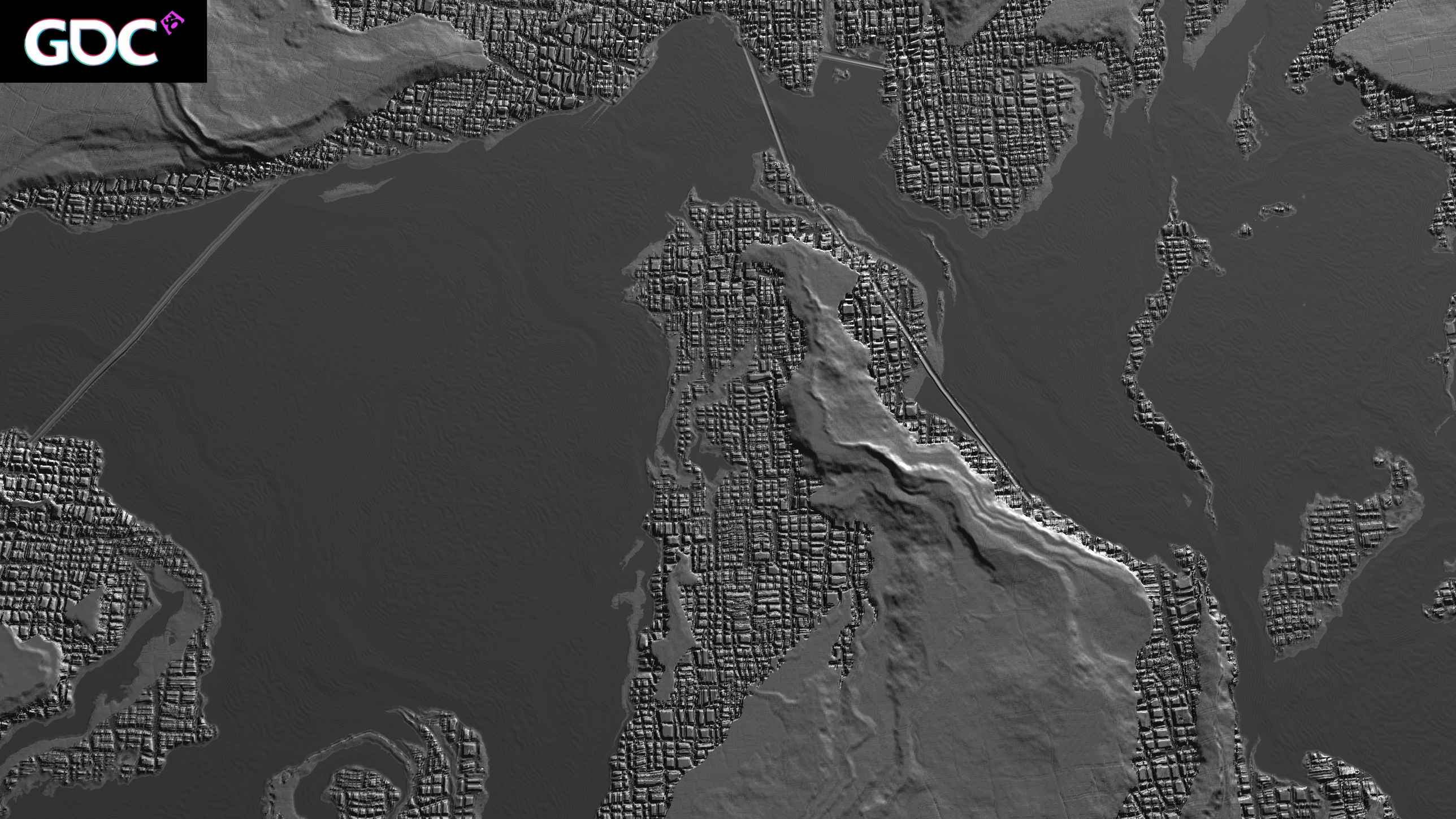


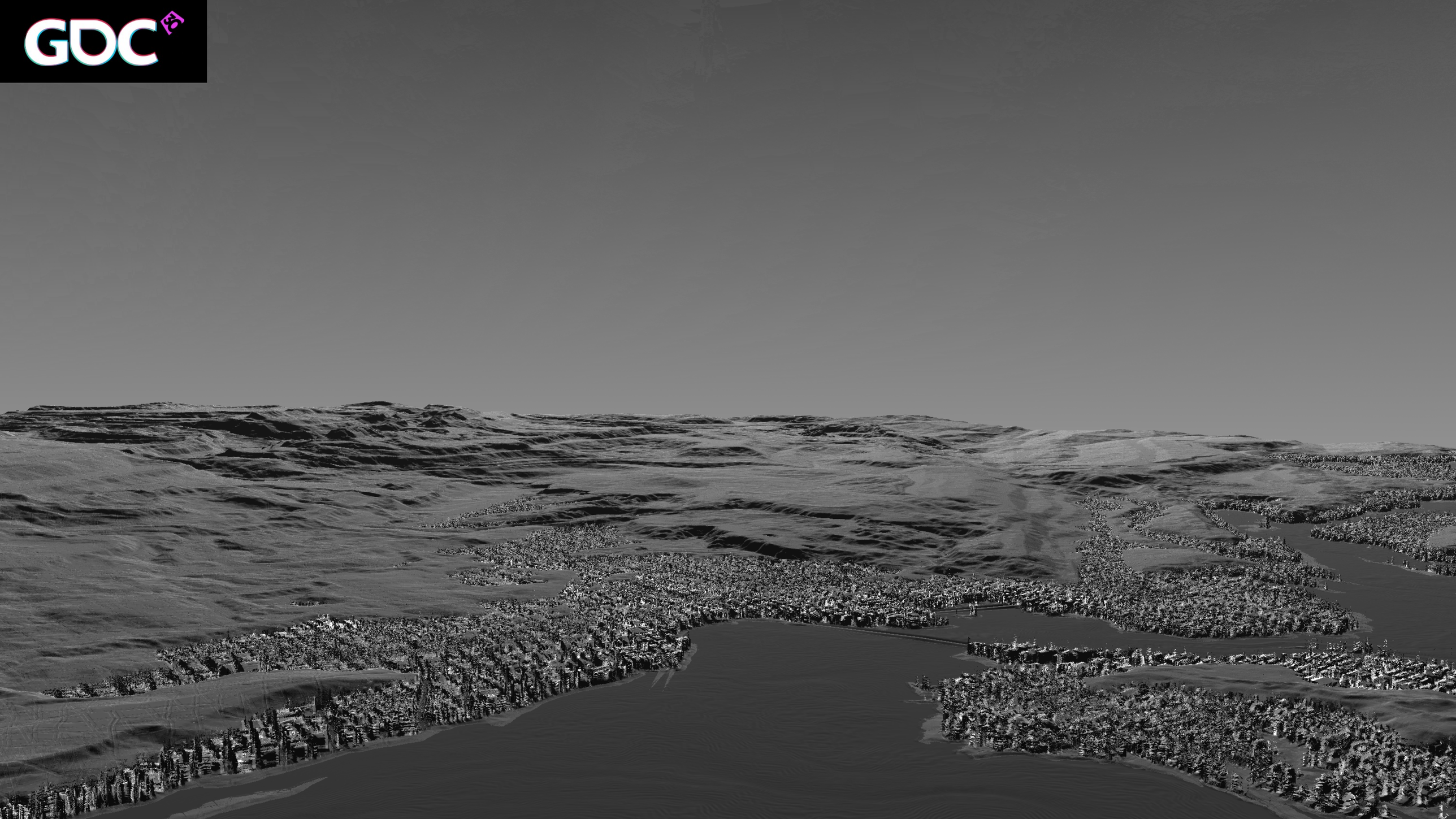
```
float noise(in vec2 uv)
{
        const float k           = 257.;
        vec4 l                  = vec4(floor(uv),fract(uv));
        l.zw                    = l.zw*l.zw*(3.-2.*l.zw);

        float u                 = l.x + l.y * k;

        vec4 v                  = vec4(u, u+1.,u+k, u+k+1.);
        v                       = fract(fract(1.23456789*v)*v/.987654321);


        l.x                     = mix(v.x, v.y, l.z);
        l.y                     = mix(v.z, v.w, l.z);
        return mix(l.x, l.y, l.w);
}



float fractal_brownian_motion(vec2 uv)
{
        float amplitude = .5;
        float frequency = 2.;

        float result  = 0.;
        for(int i = 0; i < 8; i++)
        {
                result       += noise(uv*frequency)*amplitude;
                amplitude    *= .5;
                frequency    *= 2.;
        }

        return result;
}
```



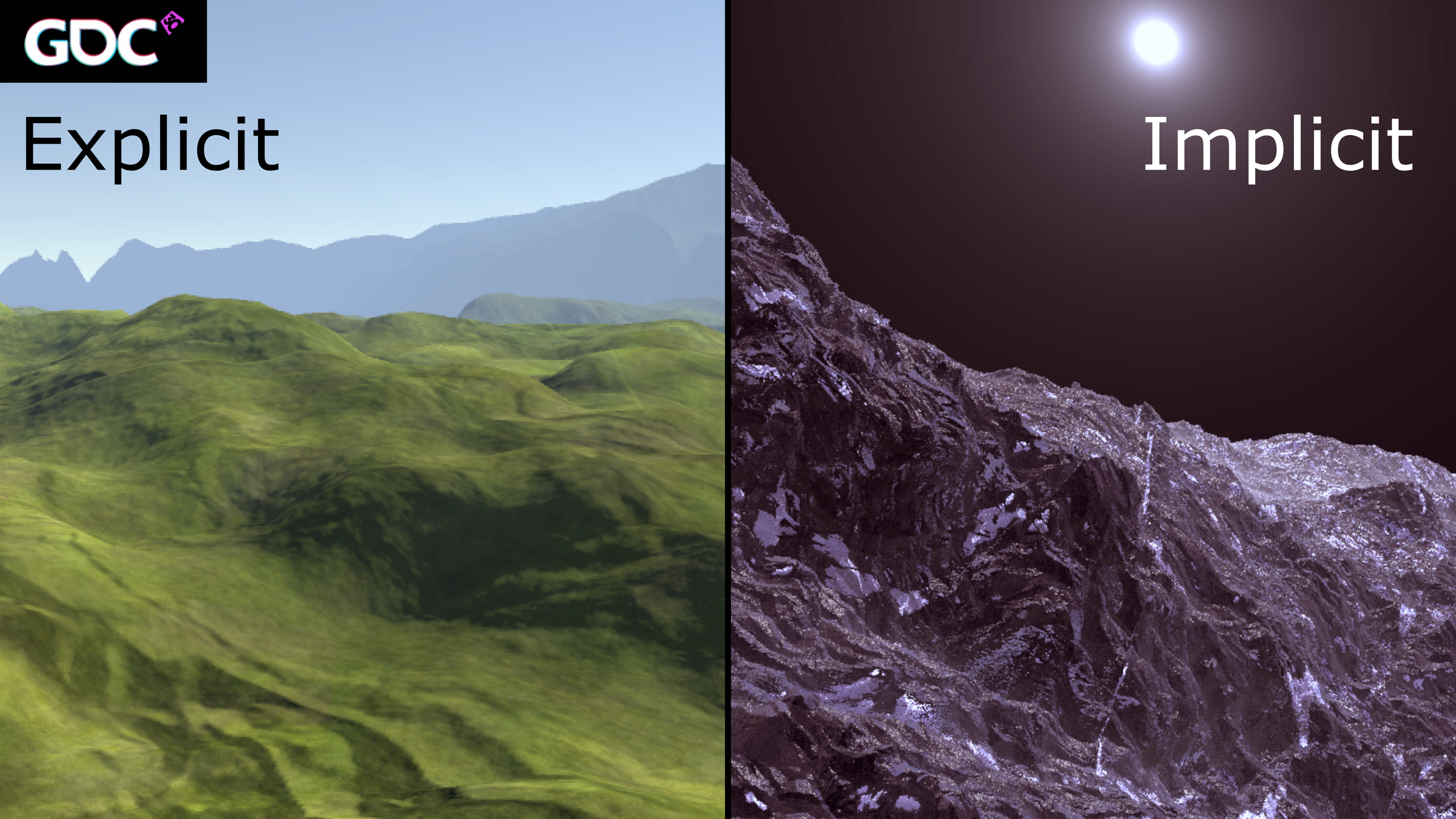



GDC

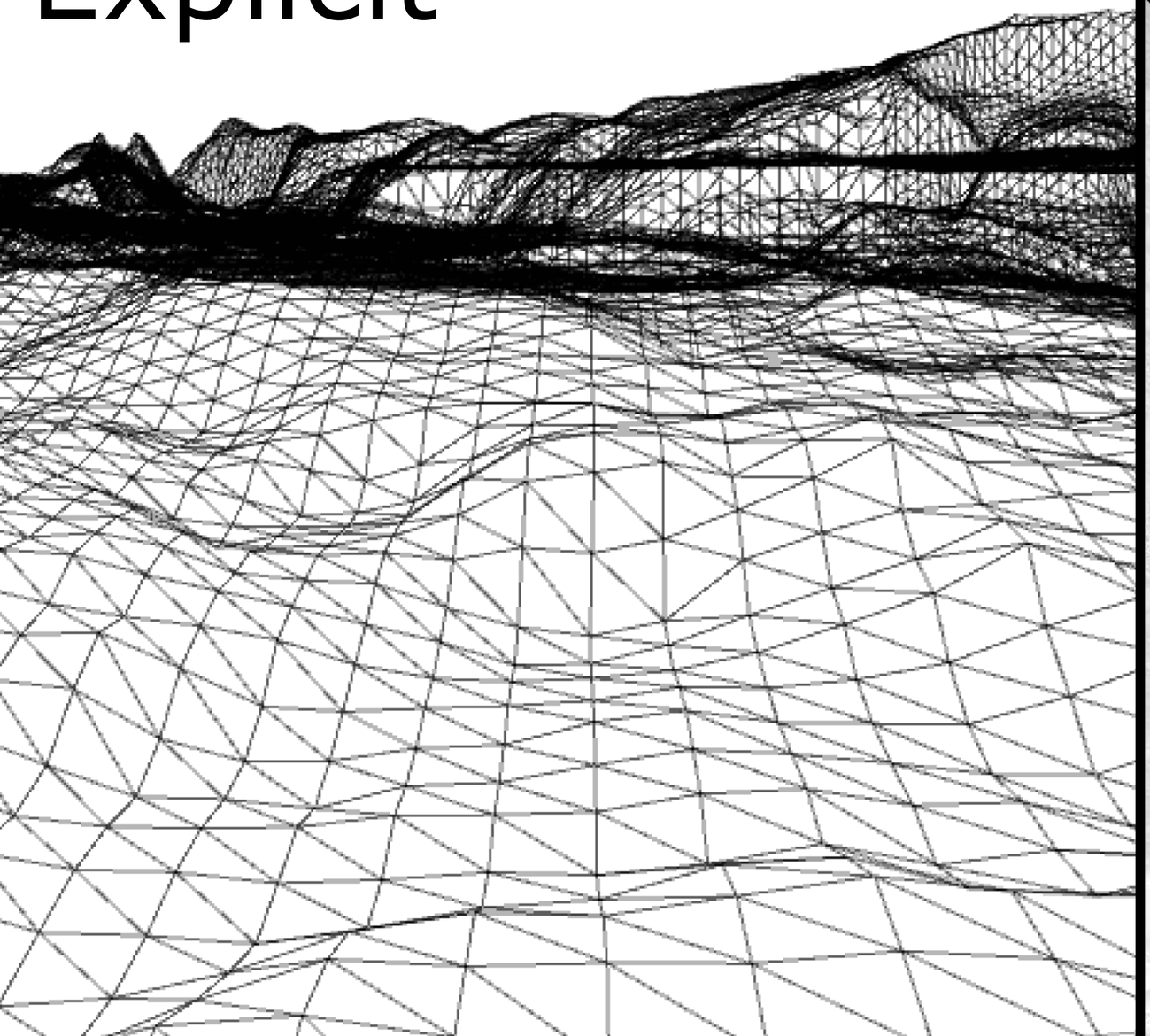

GDC



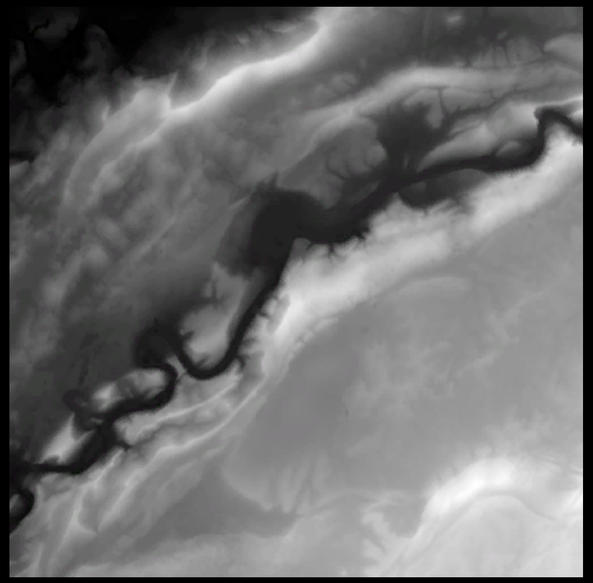
# Explicit Data vs Implicit Functions

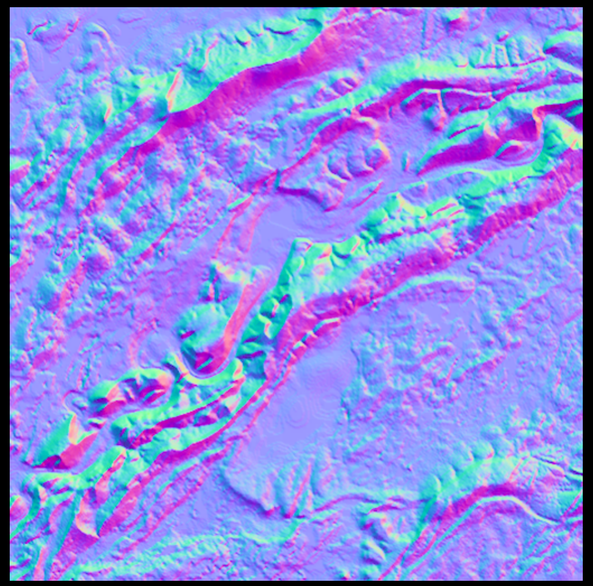

Explicit
Implicit

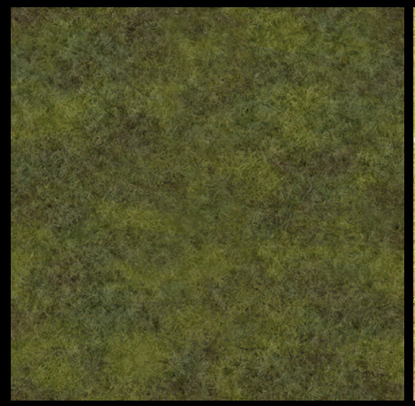
Explicit

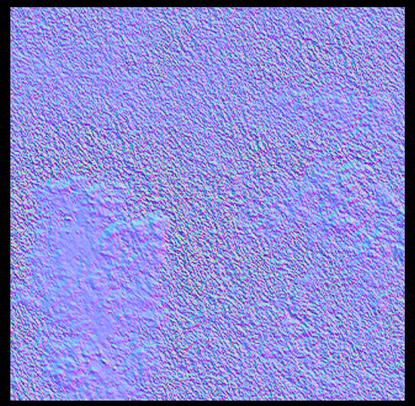


Implicit









```
float fold(in float x)
{
            return abs(fract(x)-.5);
}


vec3 fold(in vec3 p)
{
            float x = fold(p.x);
            return vec3(fold(p.z+fold(p.y)), fold(p.z+x), fold(p.y+x));
}


float fold_harmonic(in vec3 p, const int it)
{

            vec3 bp = abs(p);

            float rz = 1.;
            for (int i = 0; i < 32; i++)
            {
                        if(i>it) break;
                        p               *= fold(bp);
                        bp              += abs(fract(bp))-rz;
                        rz              += dot(fold(p), bp)-rz;

            }
            return rz*float(it);
}
```

# Explicit Data

# Explicit Data

Data is stored in memory as independent values.

# Explicit Data

Data is stored in memory as independent values.

Mesh vertices, texture pixels, etc…

# Explicit Data

Data is stored in memory as independent values.

Mesh vertices, texture pixels, etc…

Data is read from memory.

# Implicit Functions

# Implicit Functions

Code is data.

# Implicit Functions

Code is data.

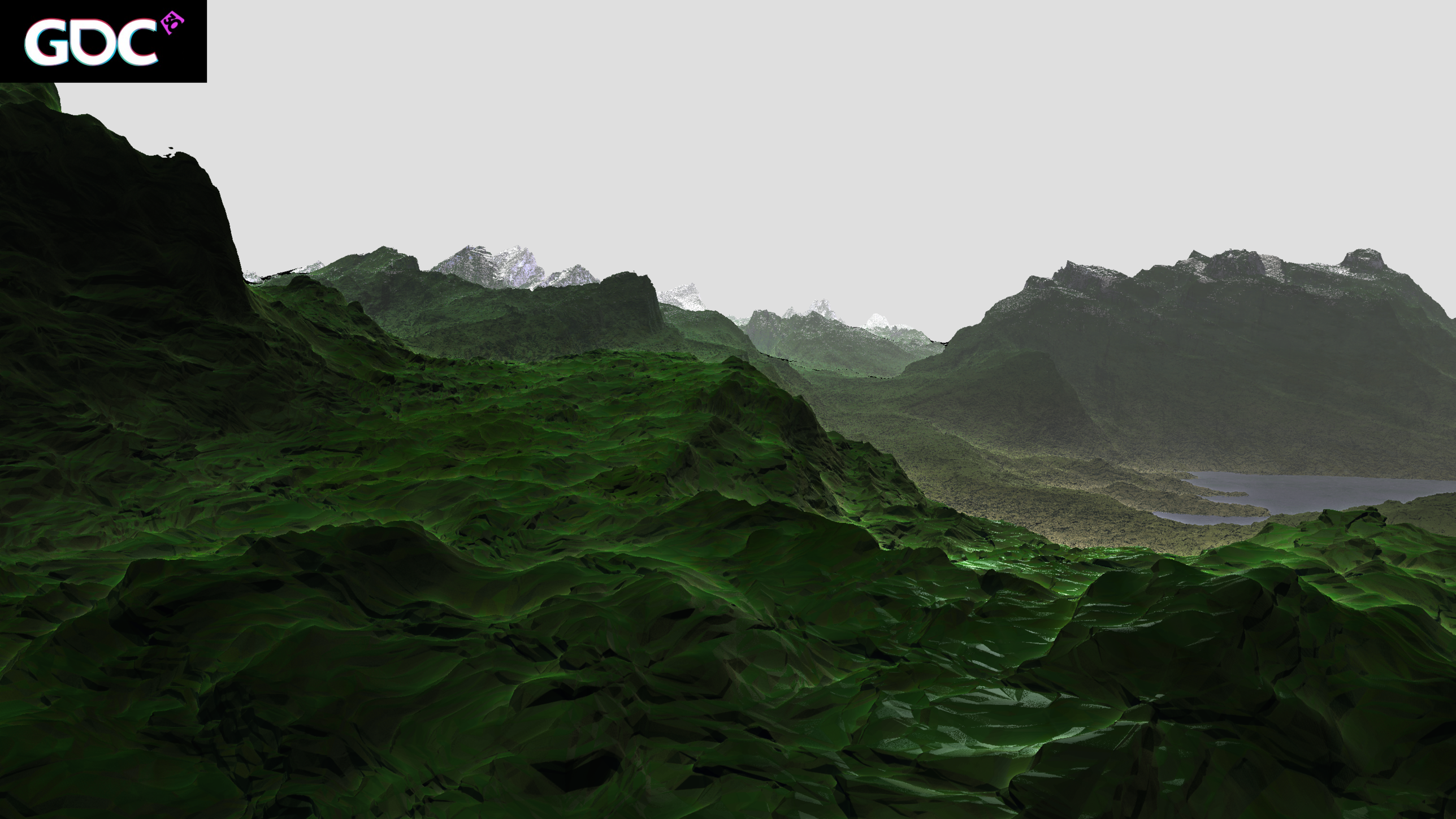
Everything is procedural.

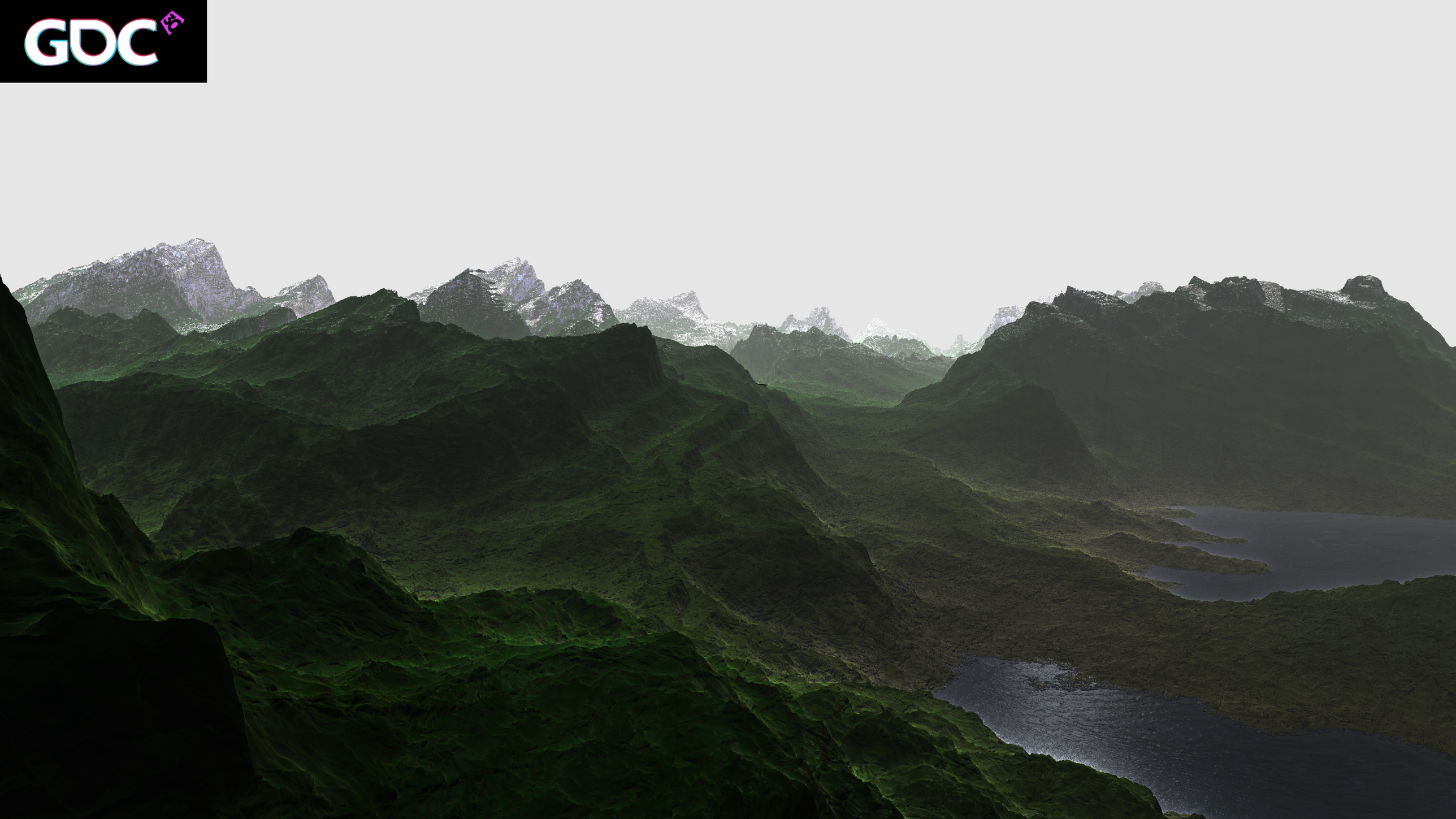
# Implicit Functions

Code is data.

Everything is procedural.

Data is accessed via computation.


GDC


GDC

# Sphere Tracing

# Sphere Tracing

Sphere tracing is a form of raytracing

(there are many.)

# Sphere Tracing

Sphere tracing is a form of raytracing

(there are many.)

It is ideal for implicit functions.

# Sphere Tracing

Sphere tracing is a form of raytracing

(there are many.)

It is ideal for implicit functions.

It is not a replacement for rasterization or voxels.

# Sphere Tracing

It is inefficient.

# Sphere Tracing

It is inefficient.

It is simple.

# Sphere Tracing

It is inefficient.

It is simple.

It is powerful.

Construct the view.

Trace the ray.

Determine which way the surface is facing.

Add light.

# Sphere Tracing
# 4 Steps

# Sphere Tracing
# 4 Steps

Construct the view.

# Sphere Tracing
# 4 Steps

Construct the view.

Trace the ray.

# Sphere Tracing
# 4 Steps

Construct the view.

Trace the ray.

Determine which way the surface is facing.

# Sphere Tracing
# 4 Steps

Construct the view.

Trace the ray.

Determine which way the surface is facing.

Add light.

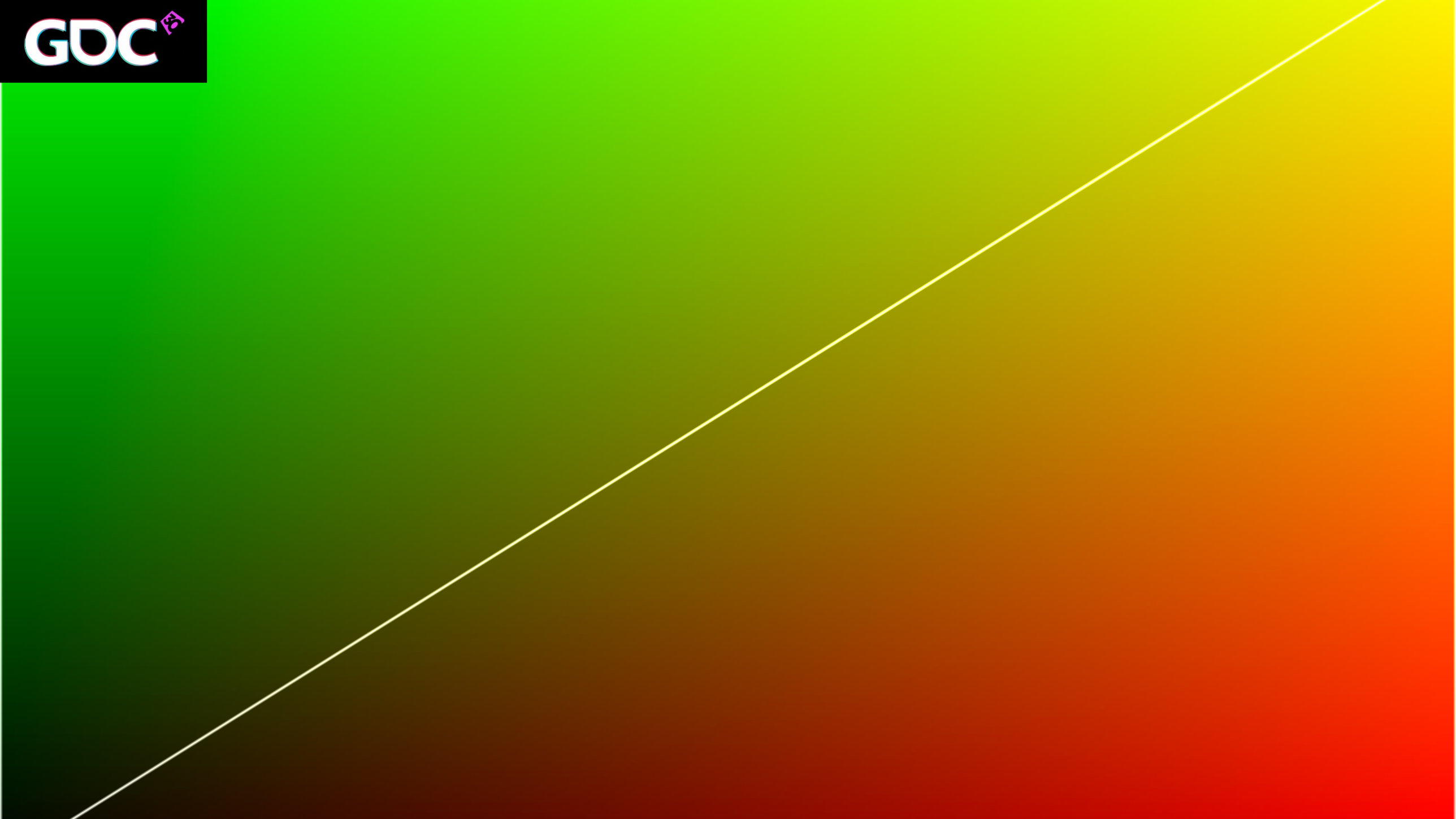
Construct the view.

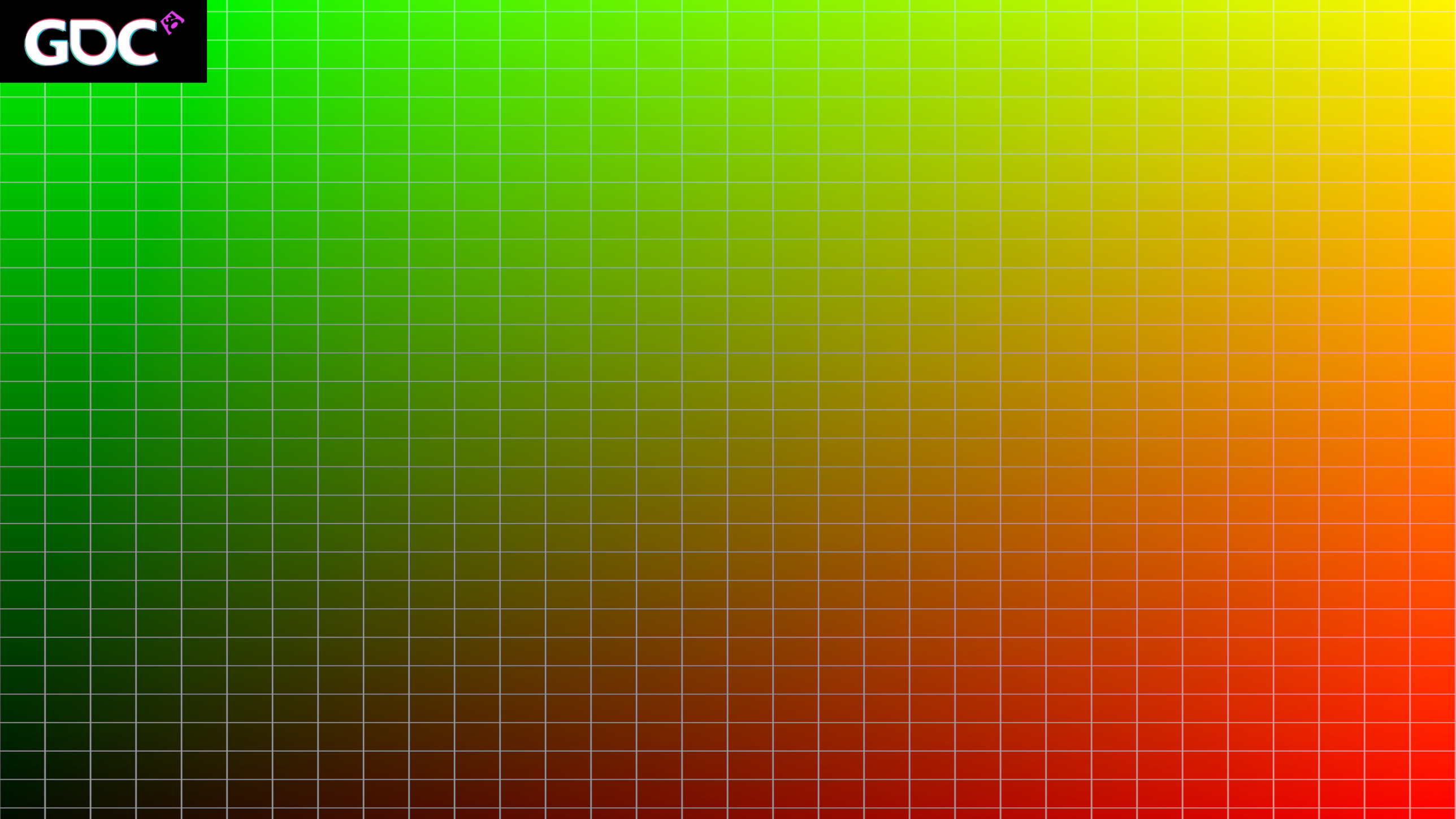
Trace the ray.

---

Determine which way the surface is facing.

Add light.

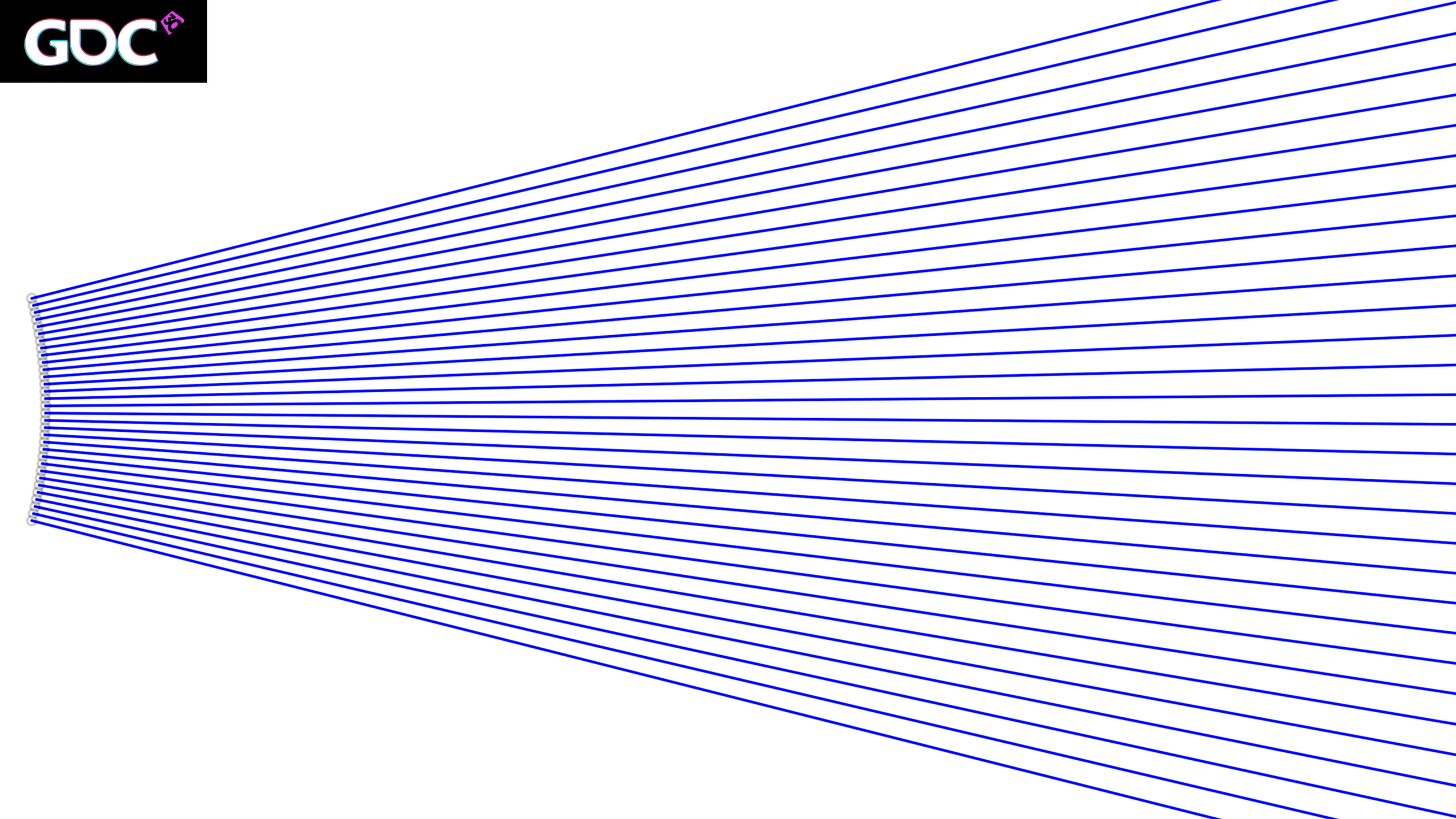

GDC


GDC

```
vec2 screen_coordinates      = gl_FragCoord.xy;
screen_coordinates           /= resolution;
screen_coordinates           = screen_coordinates - .5;
screen_coordinates           *= resolution/min(resolution.x, resolution.y);

float field_of_view          = 1.5;

vec3 direction               = vec3(screen_coordinates, field_of_view);
direction                    = normalize(direction);
```

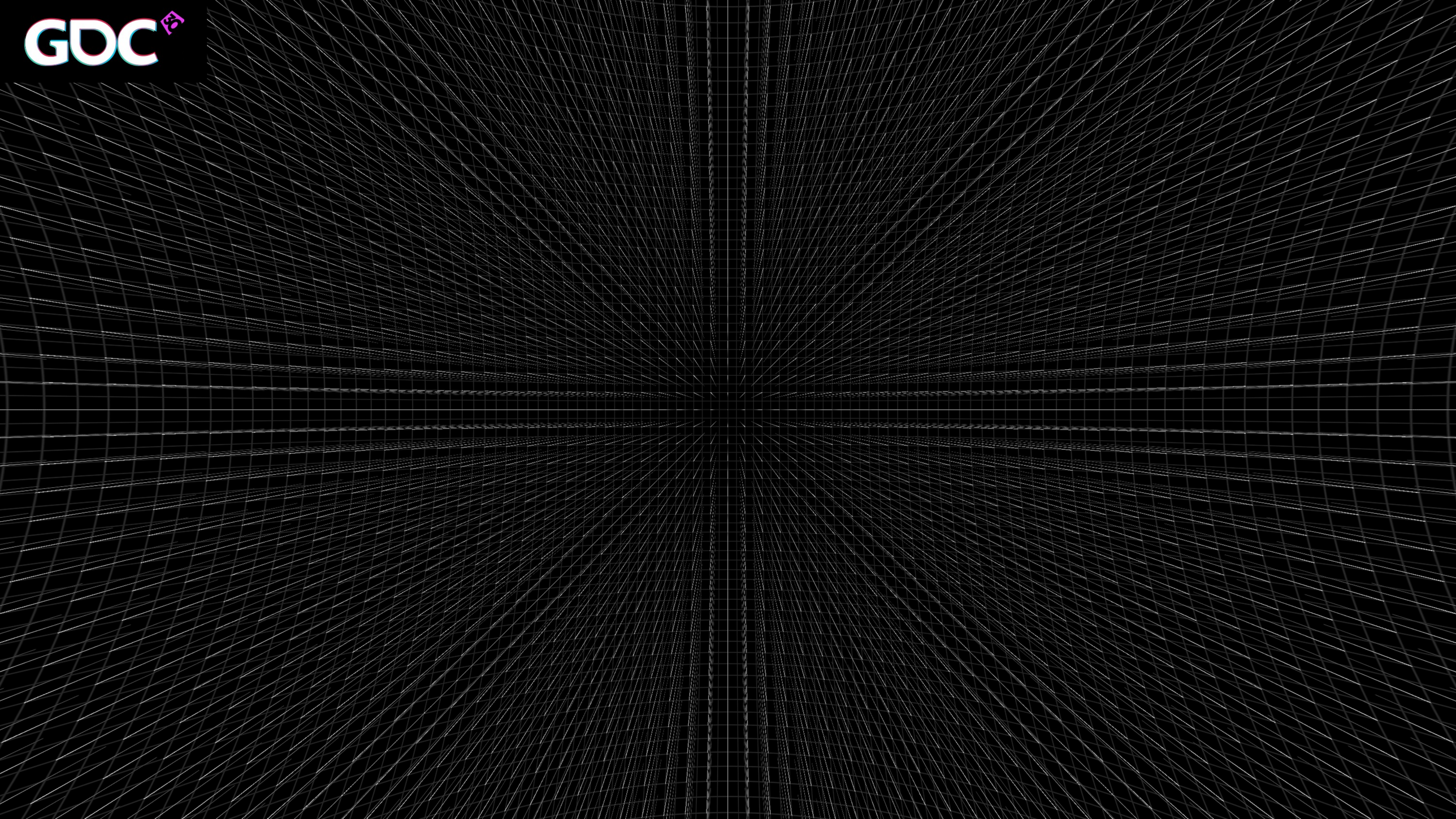
```
vec2 screen_coordinates      = gl_FragCoord.xy;
screen_coordinates           /= resolution;
screen_coordinates           = screen_coordinates - .5;
screen_coordinates           *= resolution/min(resolution.x, resolution.y);

float field_of_view          = 1.5;

vec3 direction               = vec3(screen_coordinates, field_of_view);
direction                    = normalize(direction);
```

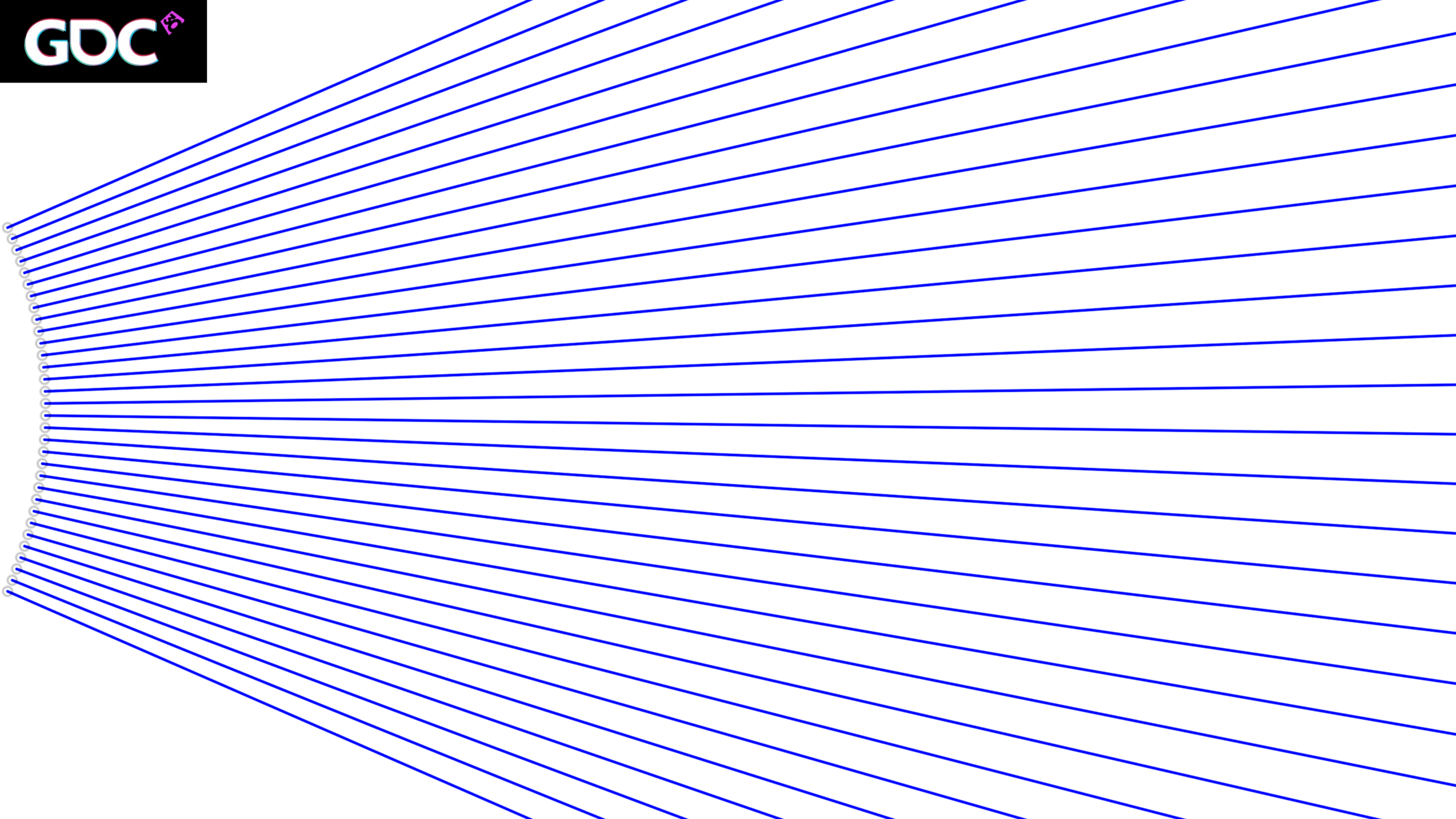
```
vec2 screen_coordinates      = gl_FragCoord.xy;
screen_coordinates           /= resolution;
screen_coordinates           = screen_coordinates - .5;
screen_coordinates           *= resolution/min(resolution.x, resolution.y);

float field_of_view          = 1.5;

vec3 direction               = vec3(screen_coordinates, field_of_view);
direction                    = normalize(direction);
```

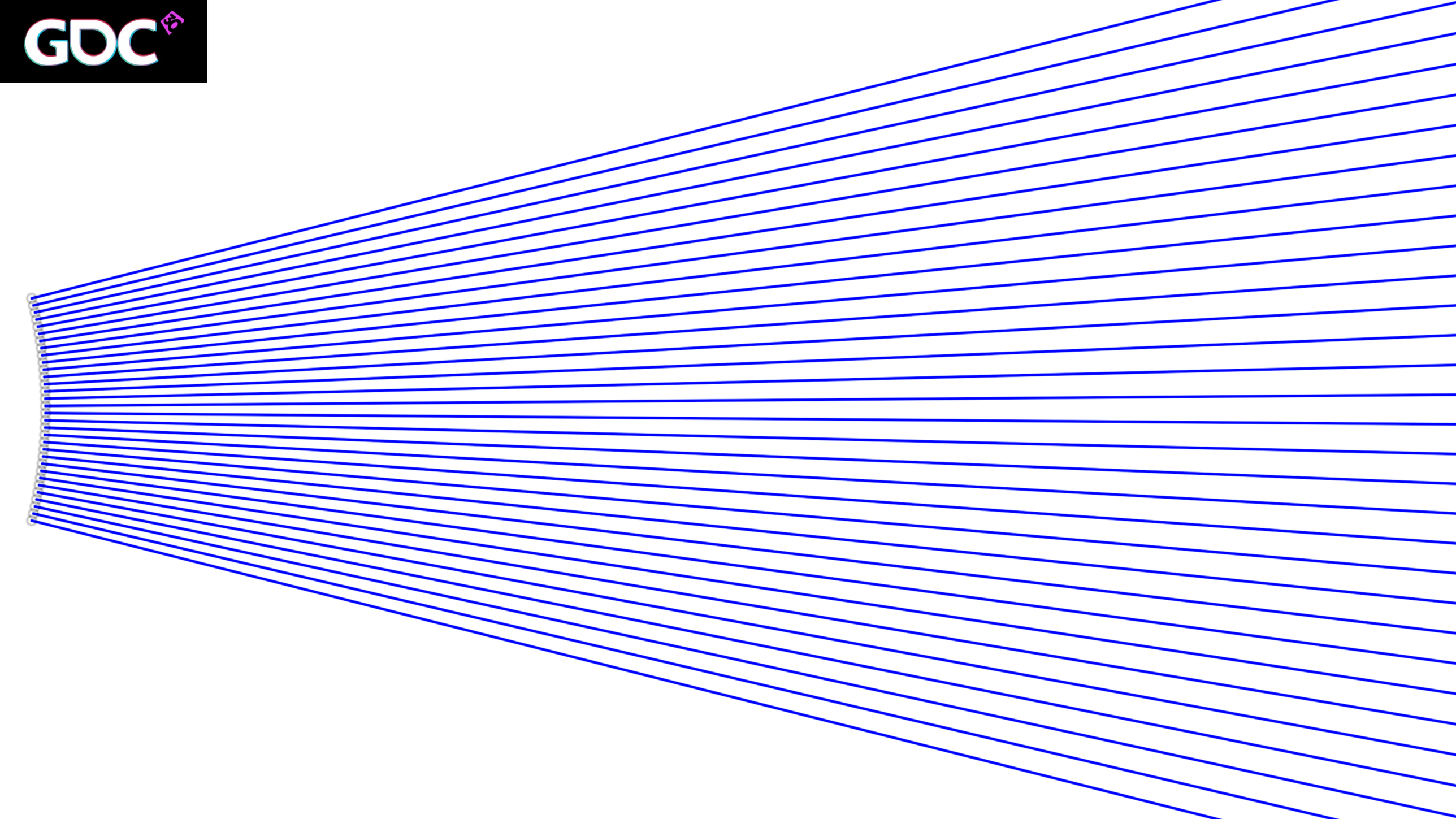


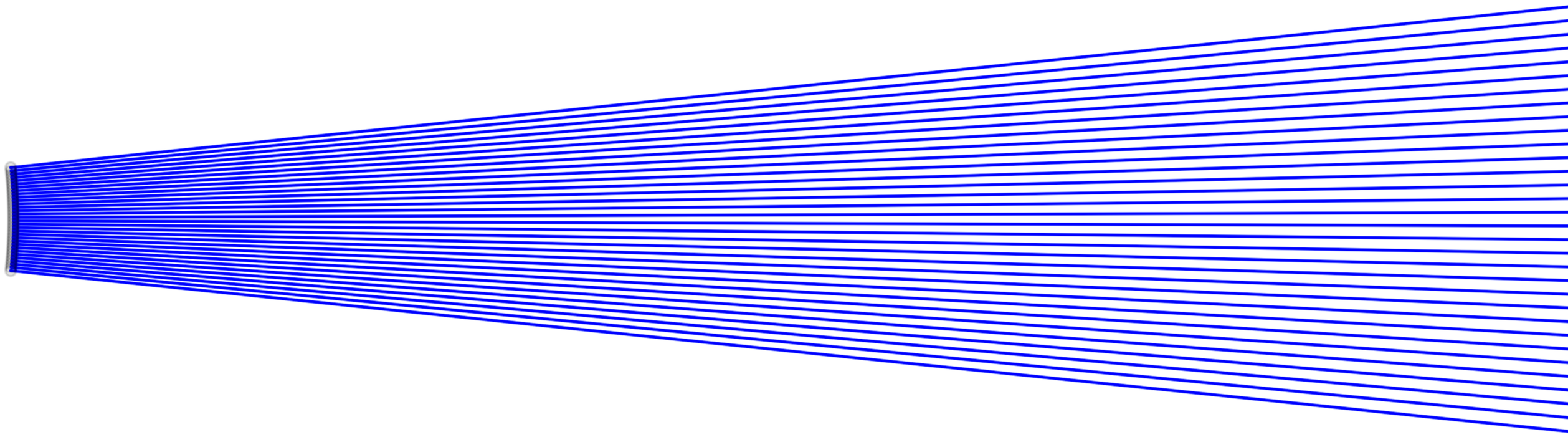
```
vec2 screen_coordinates      = gl_FragCoord.xy;
screen_coordinates           /= resolution;
screen_coordinates           = screen_coordinates - .5;
screen_coordinates           *= resolution/min(resolution.x, resolution.y);

float field_of_view          = 1.5;

vec3 direction               = vec3(screen_coordinates, field_of_view);
direction                    = normalize(direction);
```

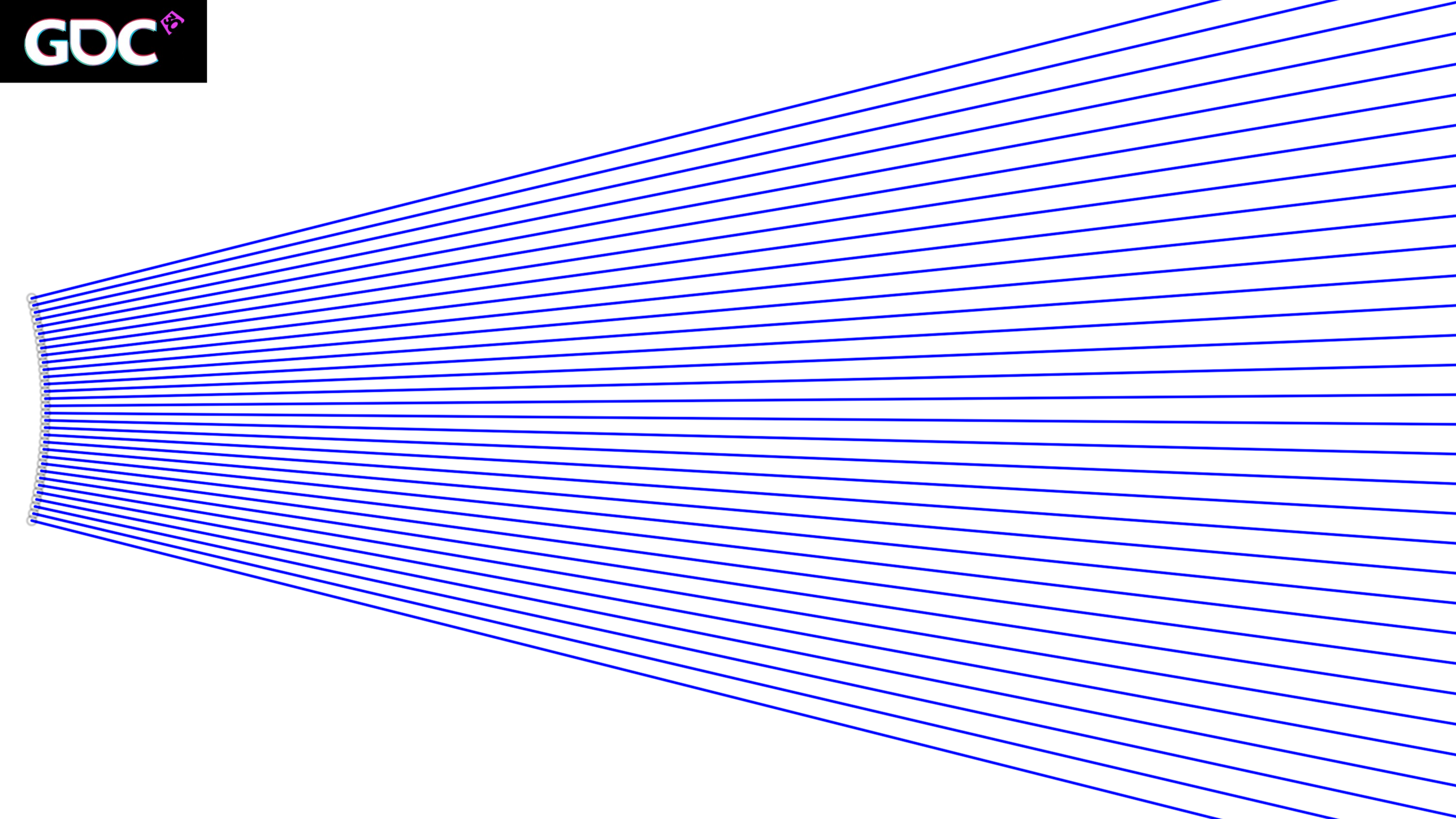
```
vec2 screen_coordinates      = gl_FragCoord.xy;
screen_coordinates           /= resolution;
screen_coordinates           = screen_coordinates - .5;
screen_coordinates           *= resolution/min(resolution.x, resolution.y);

float field_of_view          = 1.5;

vec3 direction               = vec3(screen_coordinates, field_of_view);
direction                    = normalize(direction);
```

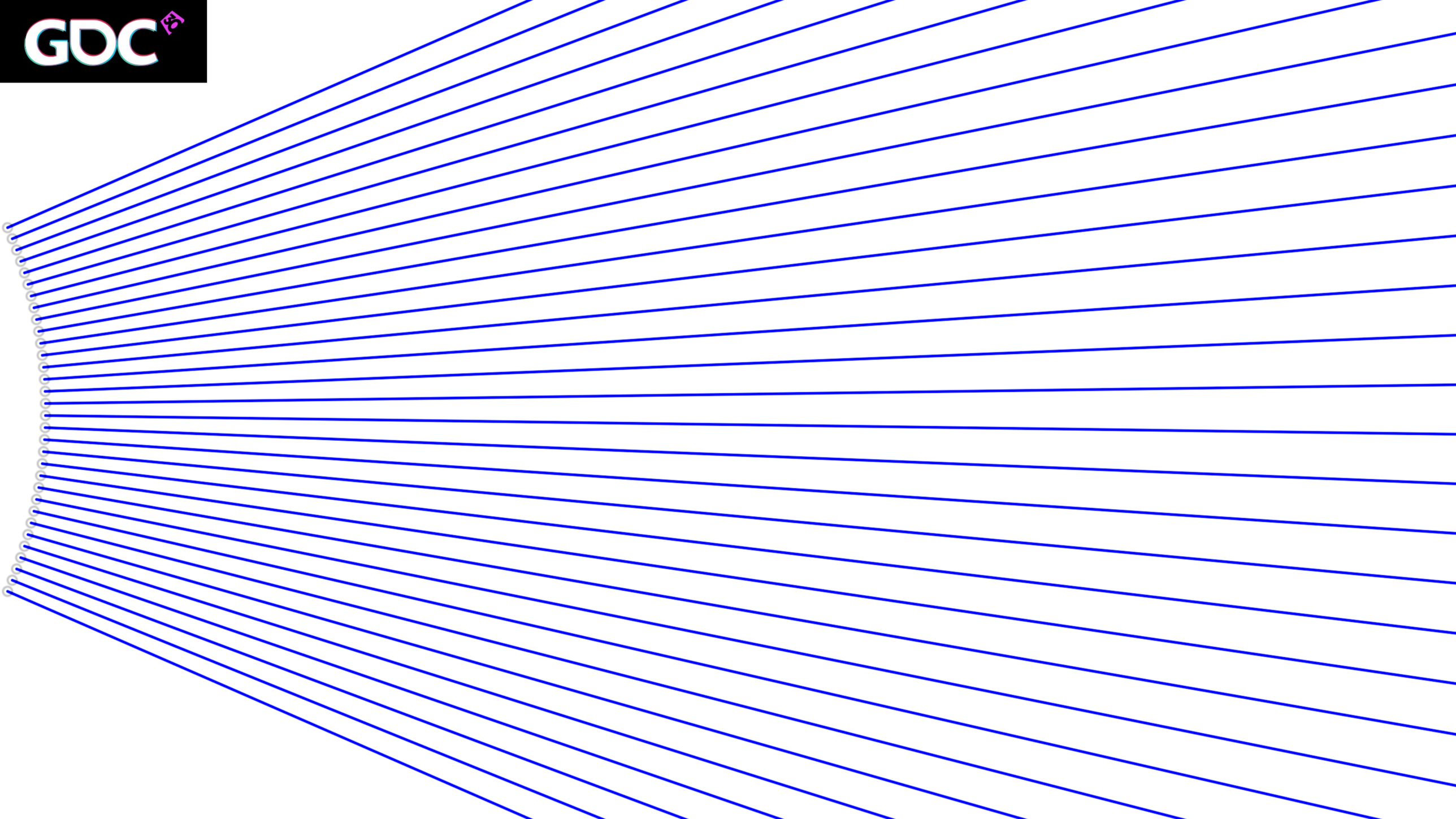




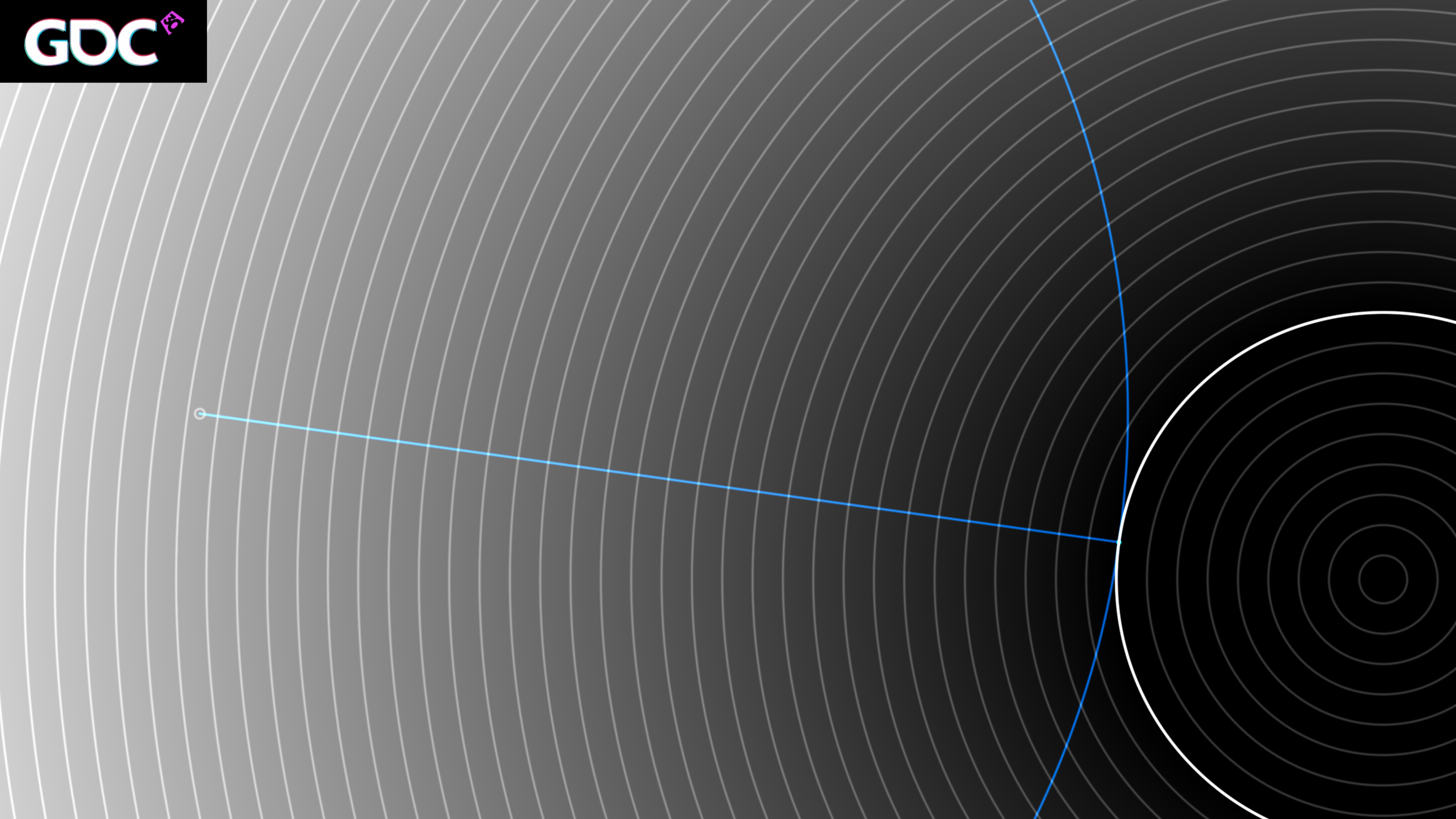


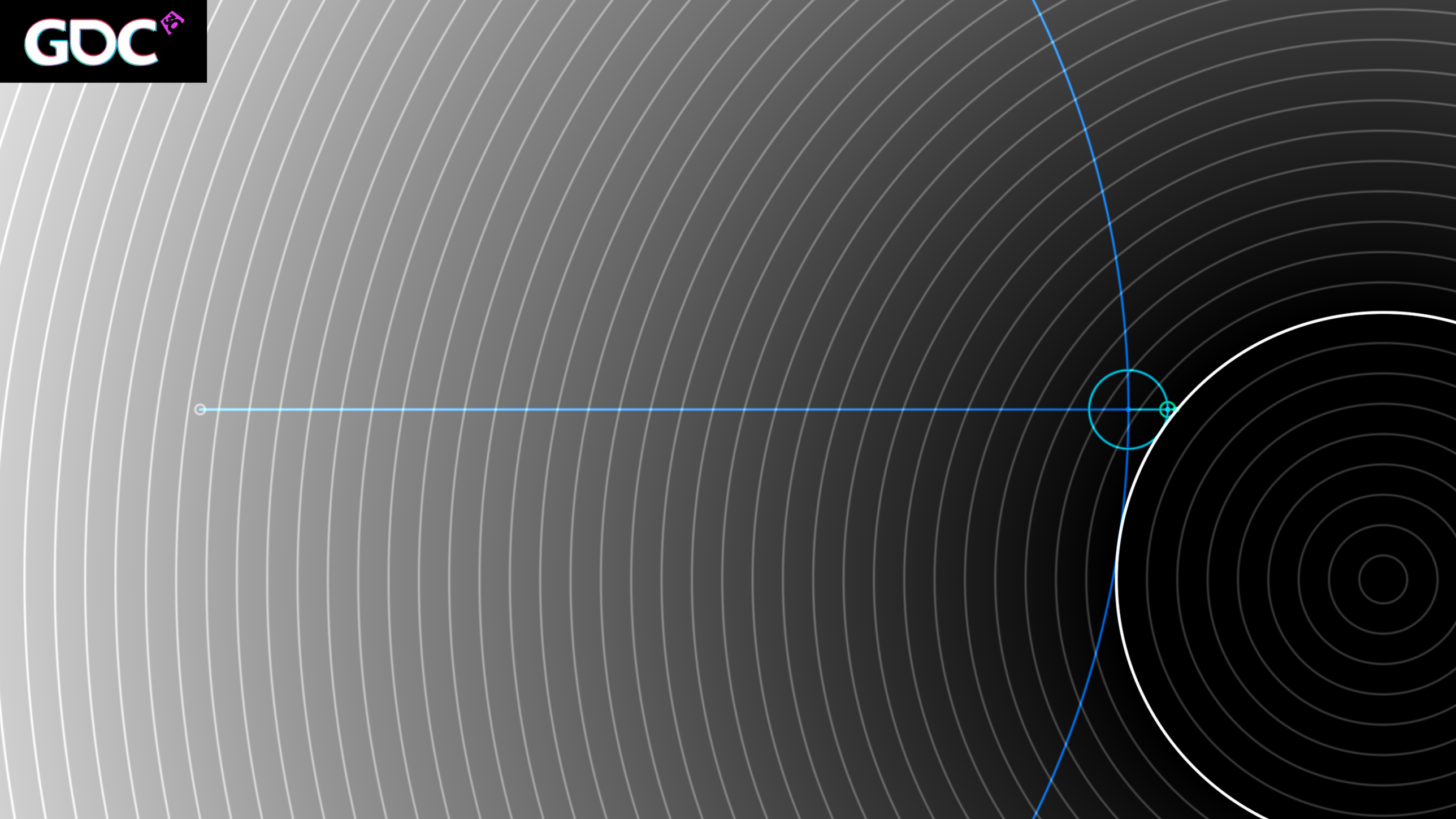


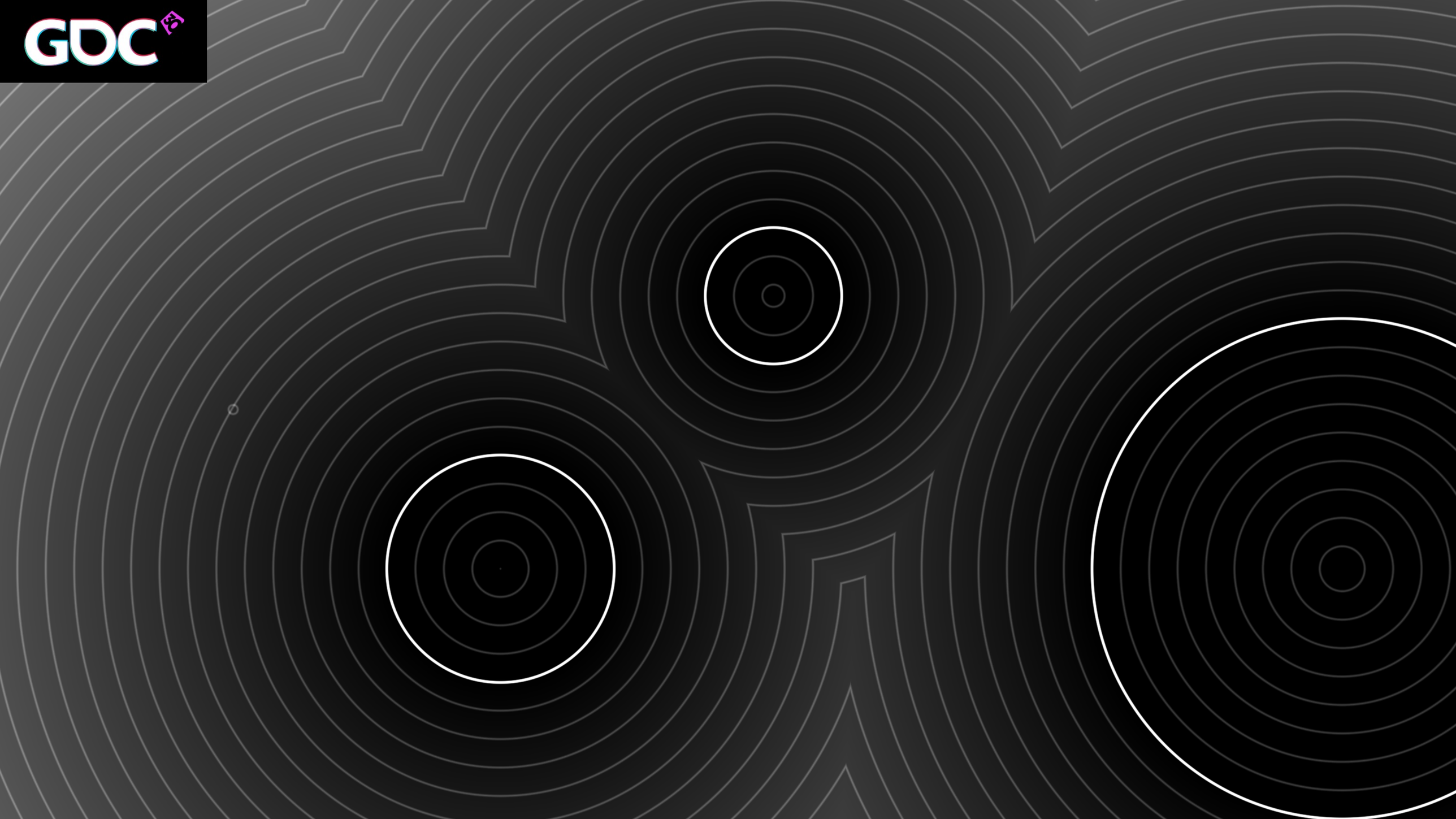


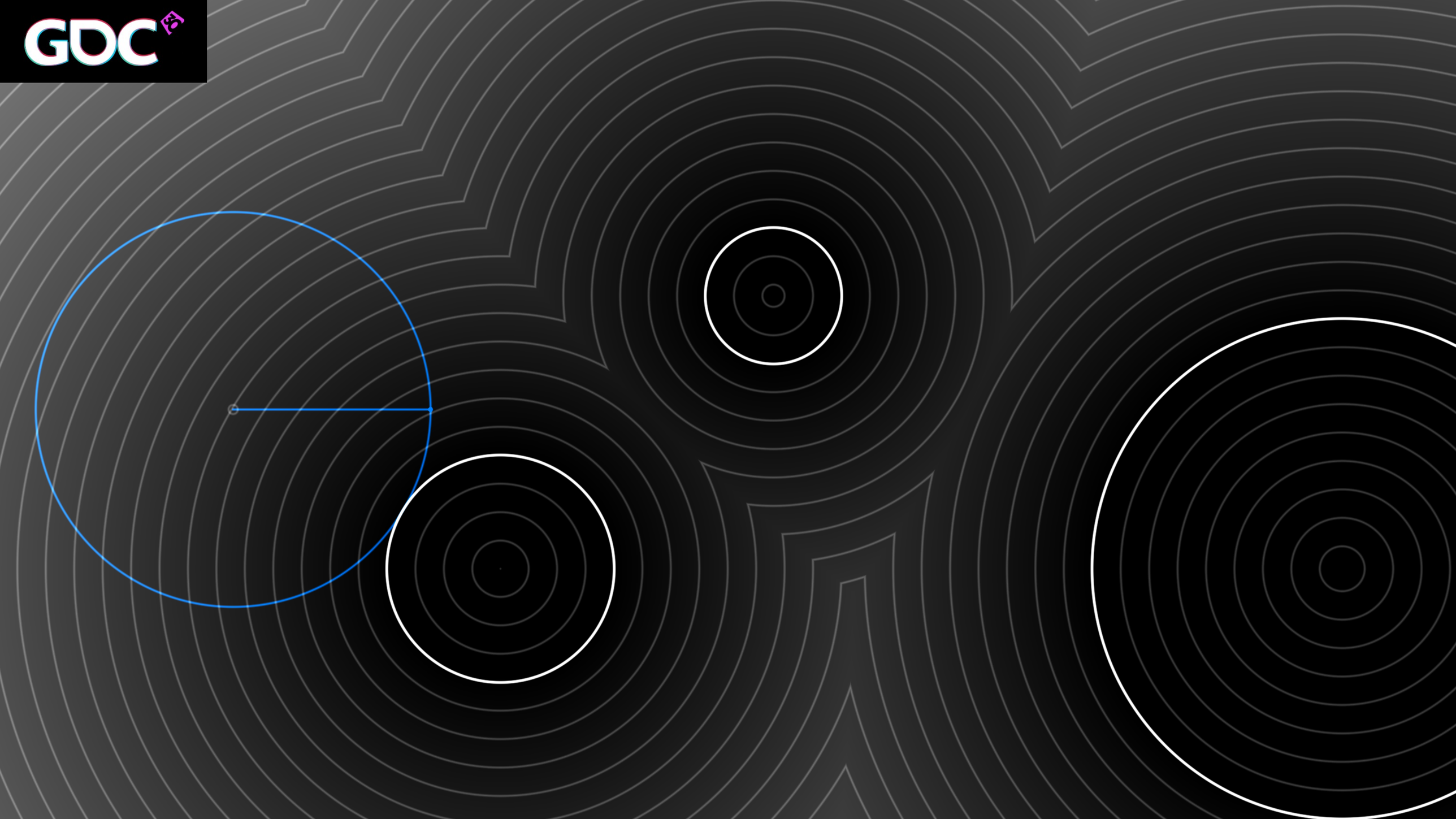


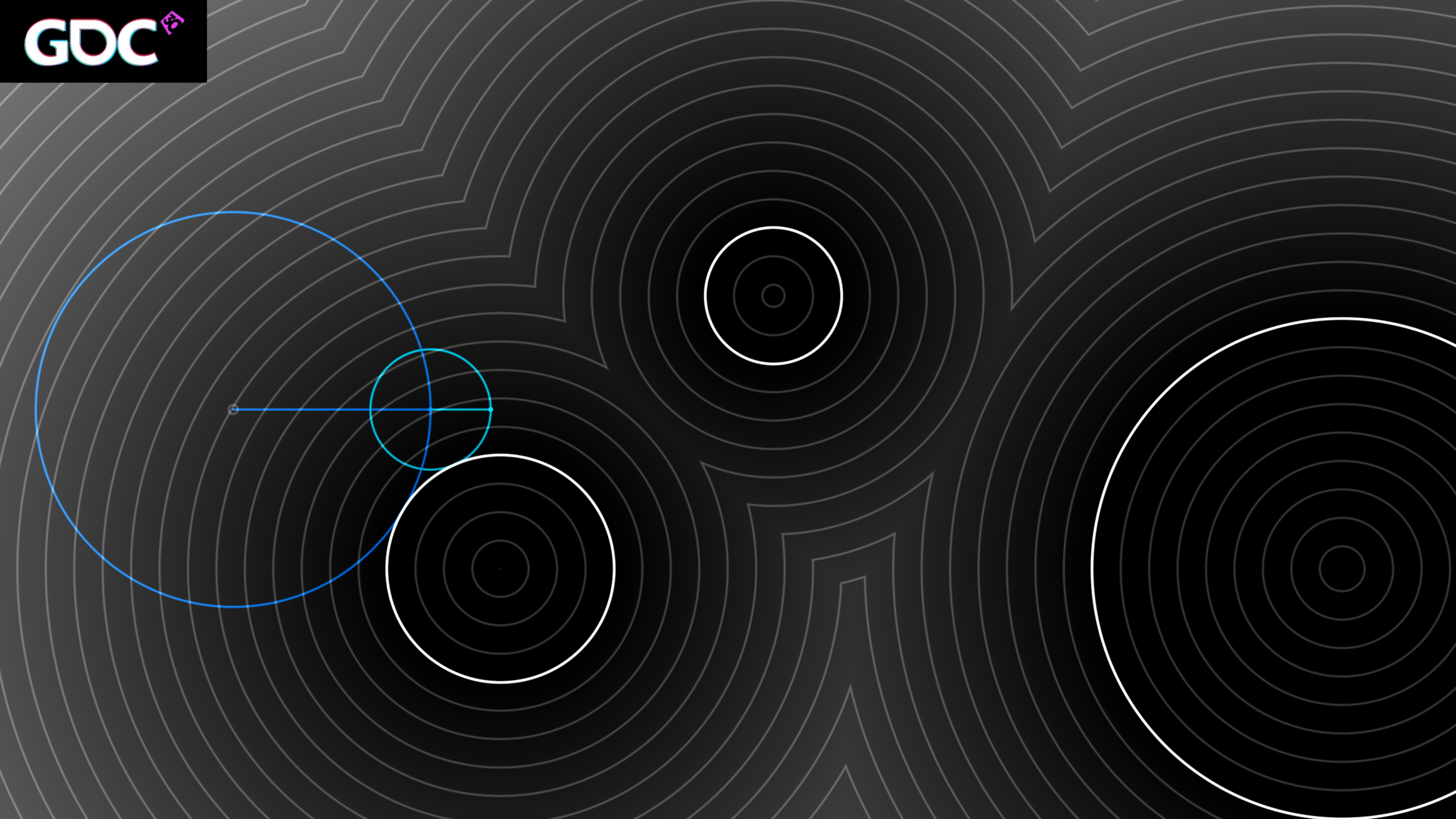
# Rays

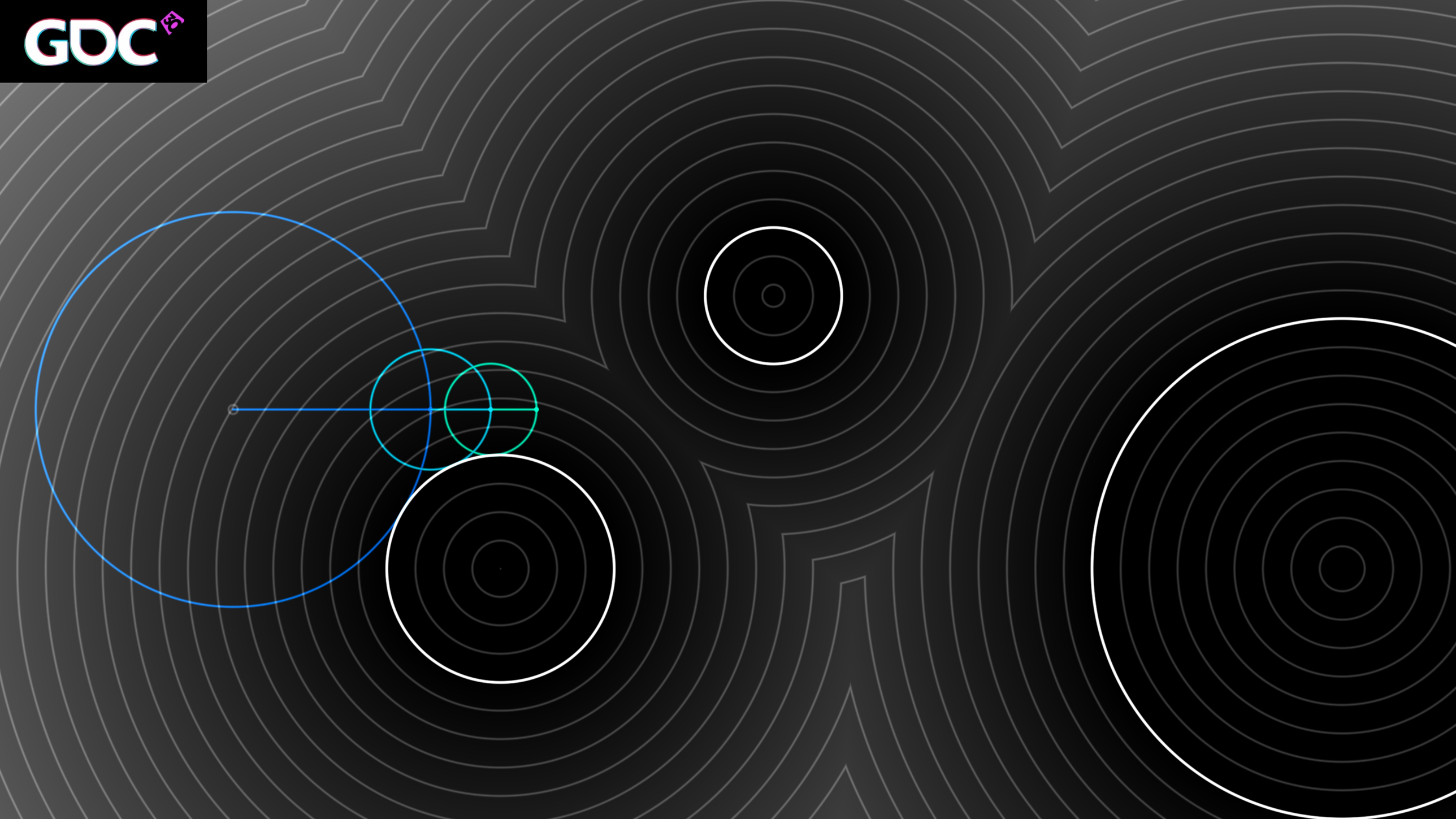


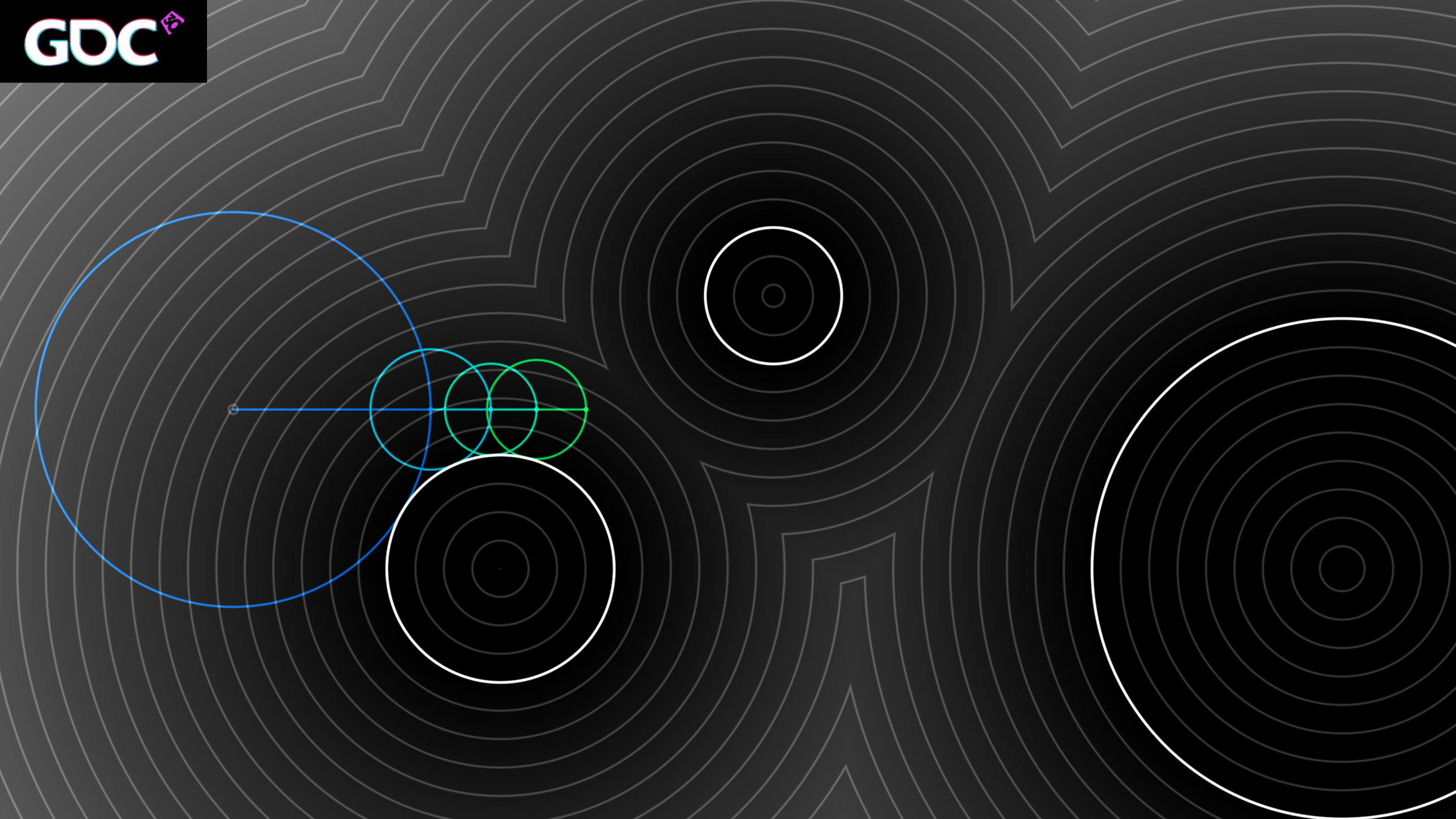


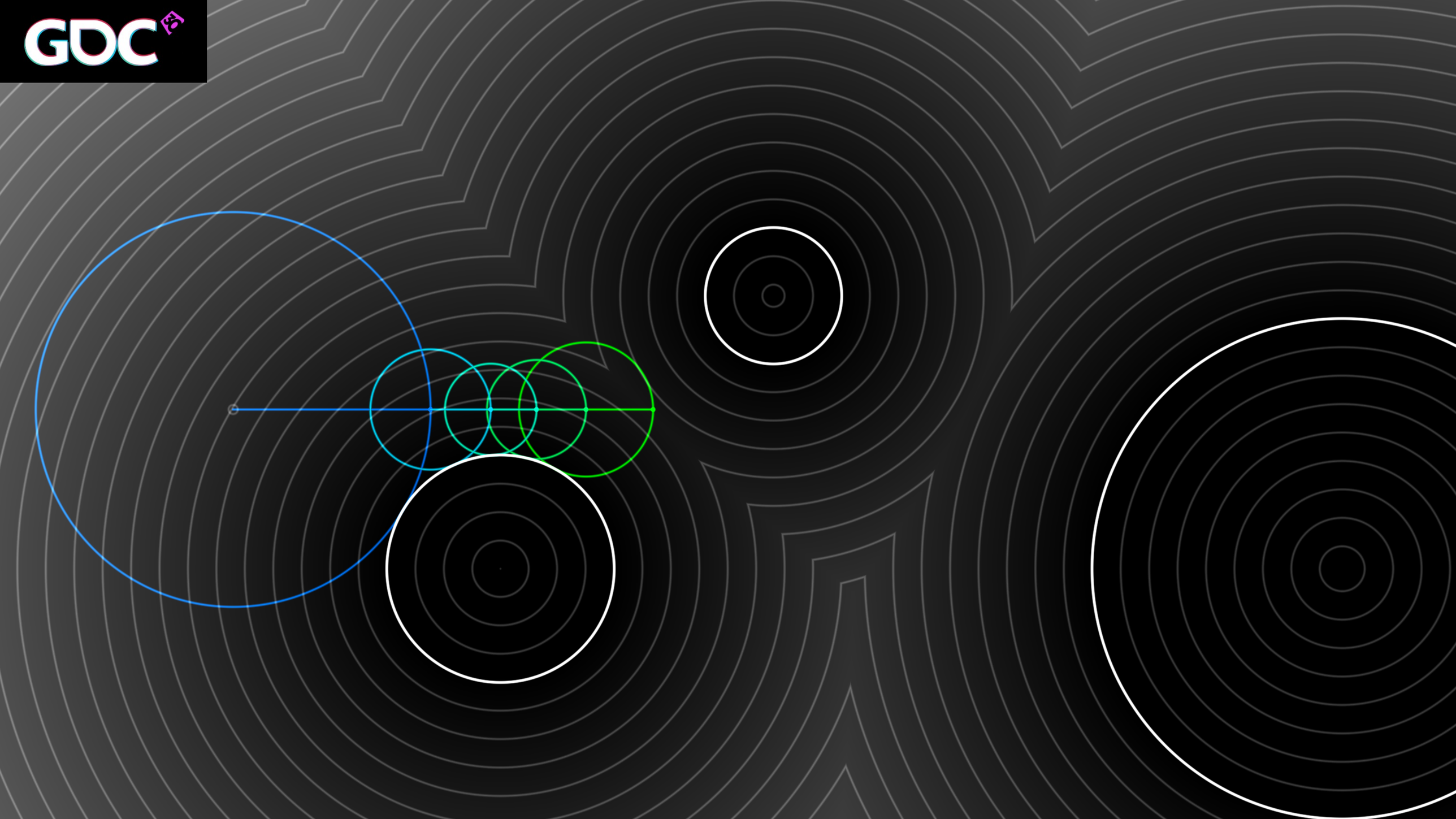

GDC

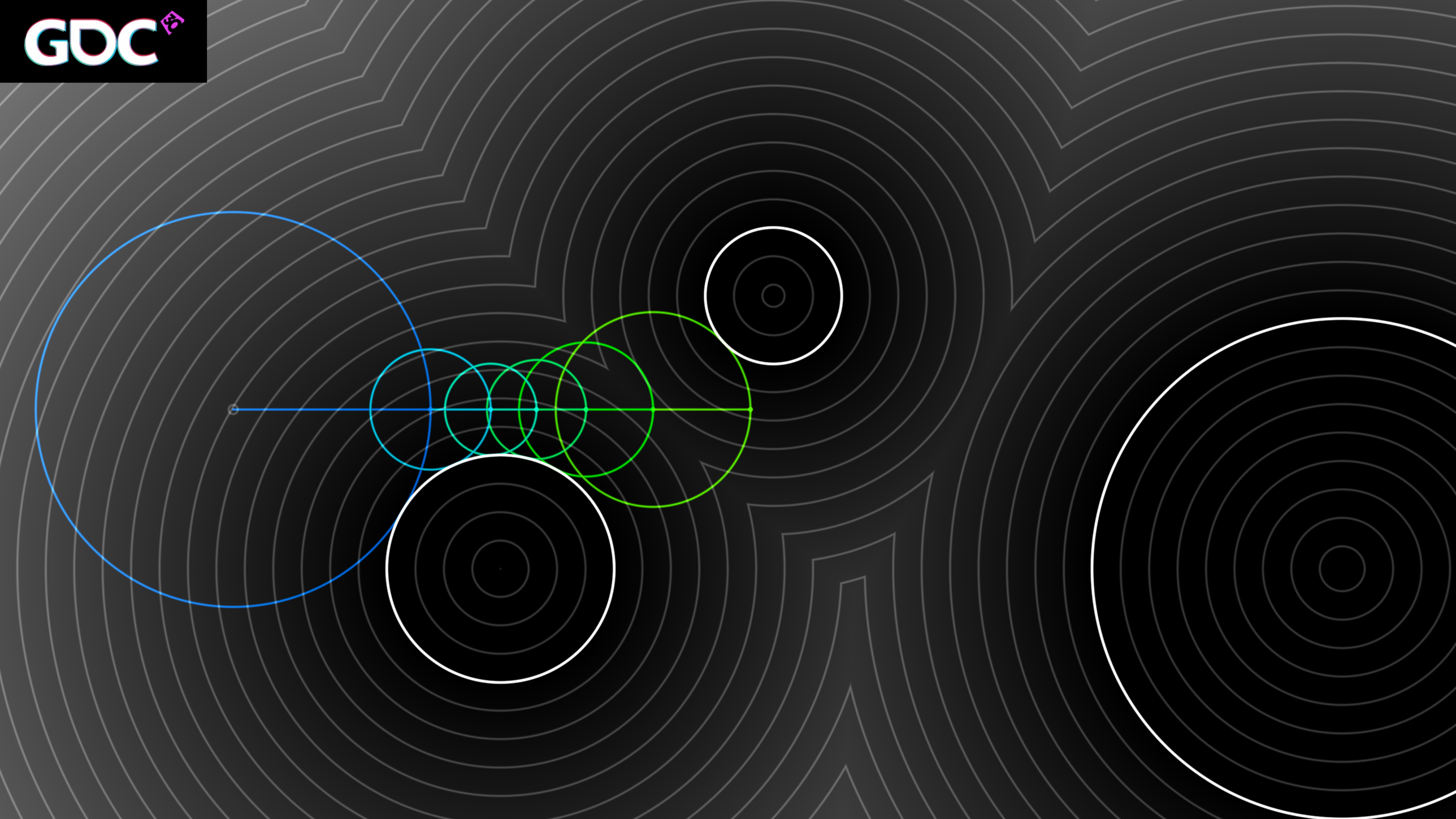


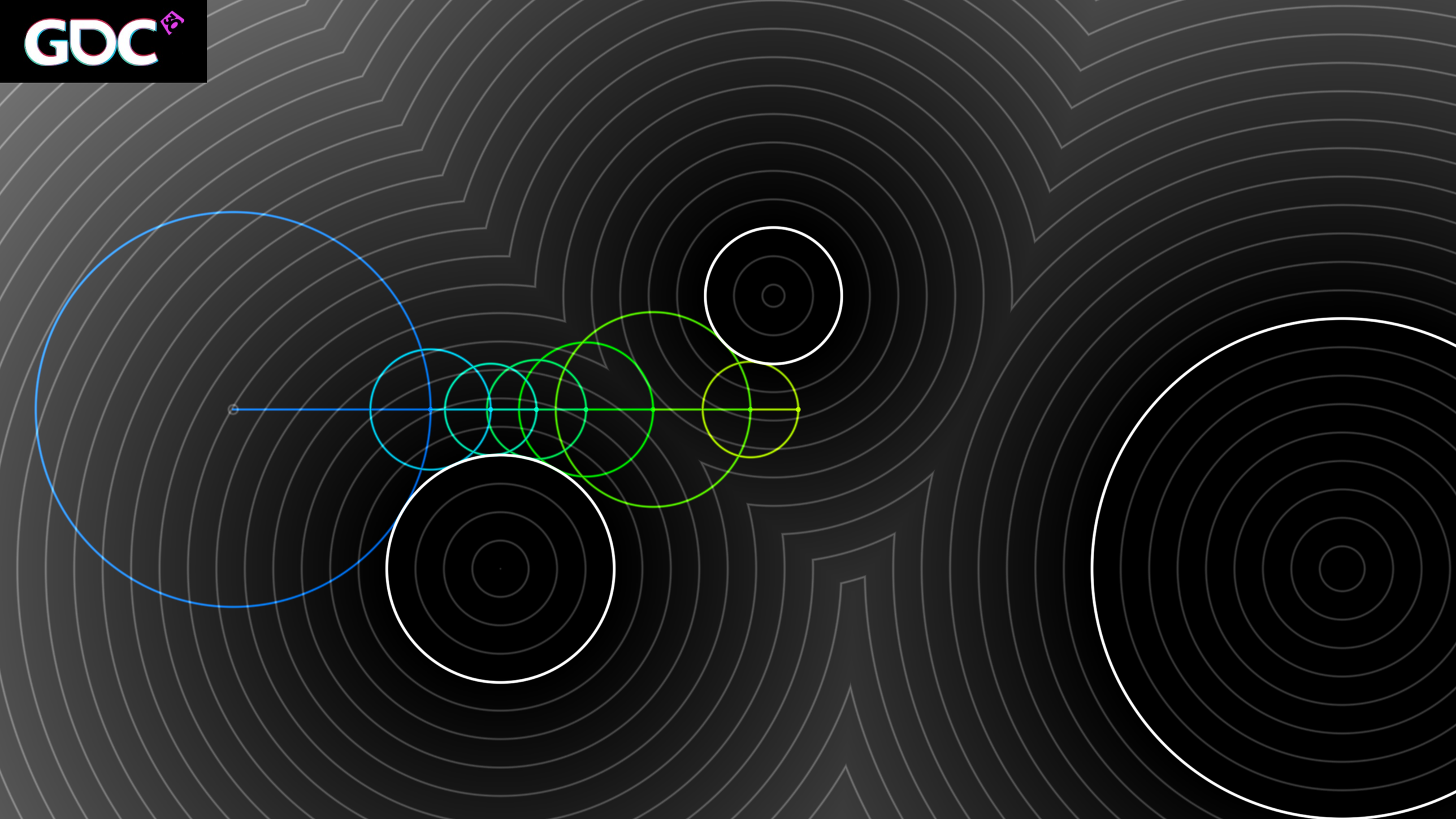


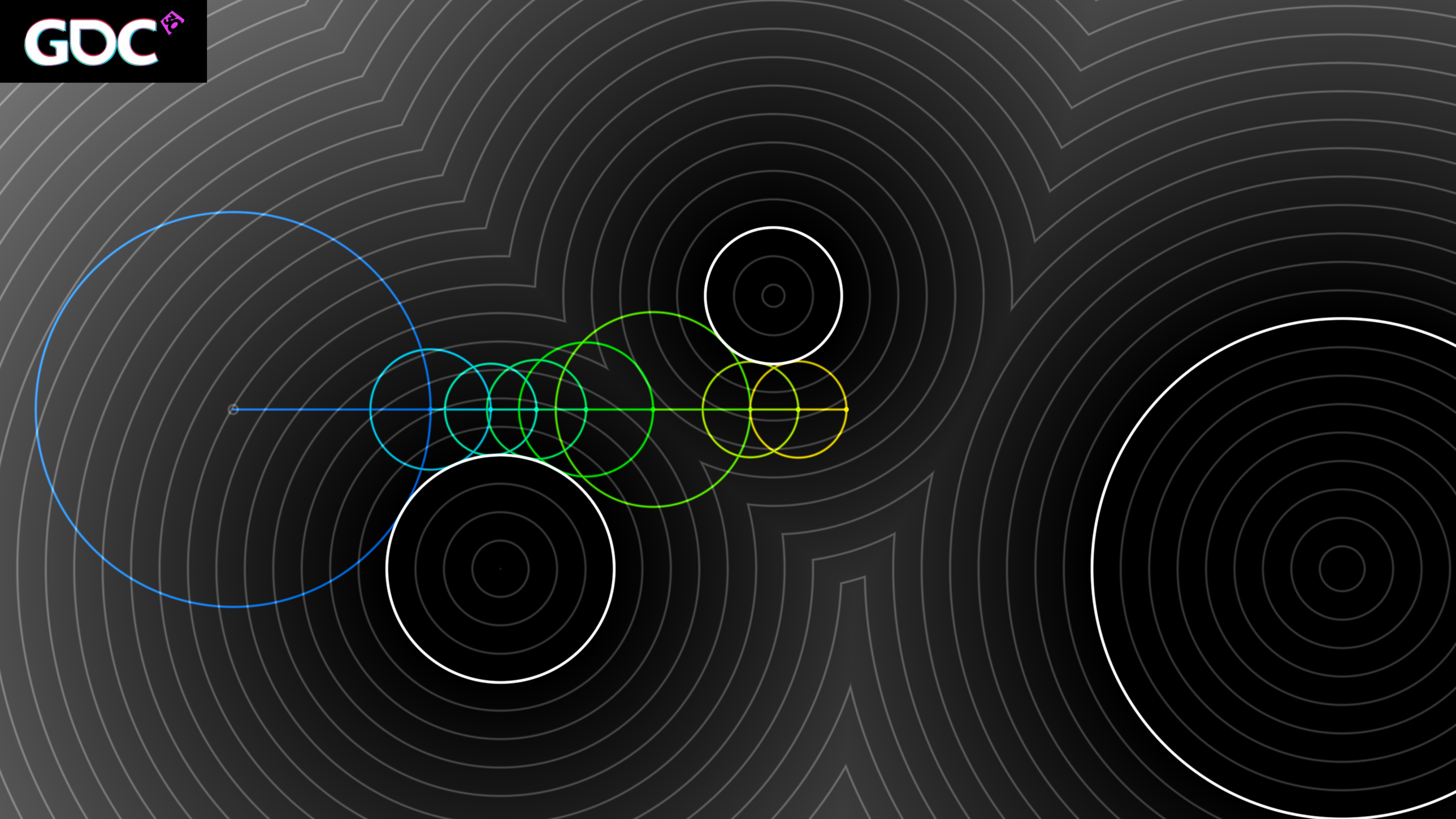


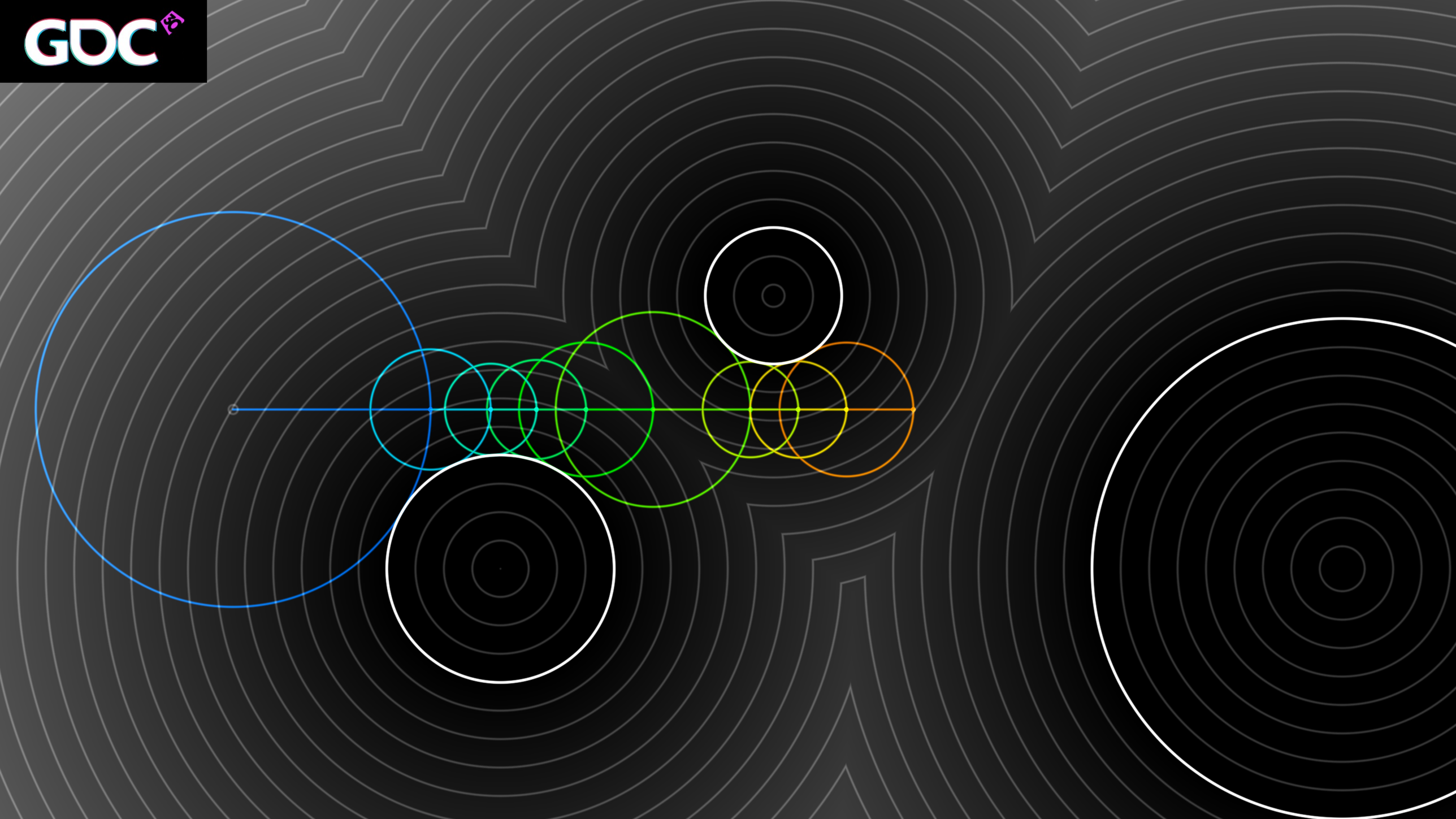


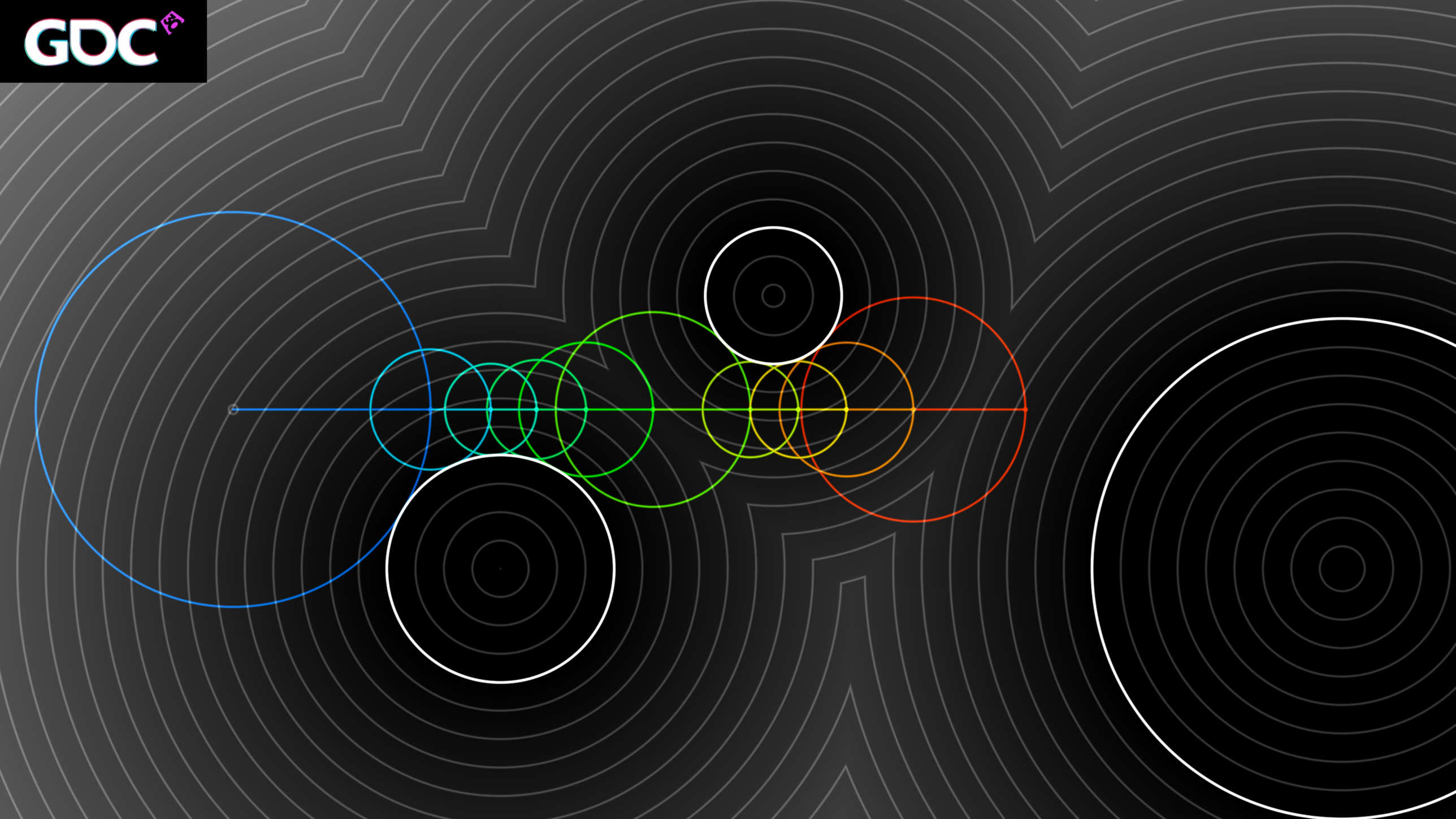


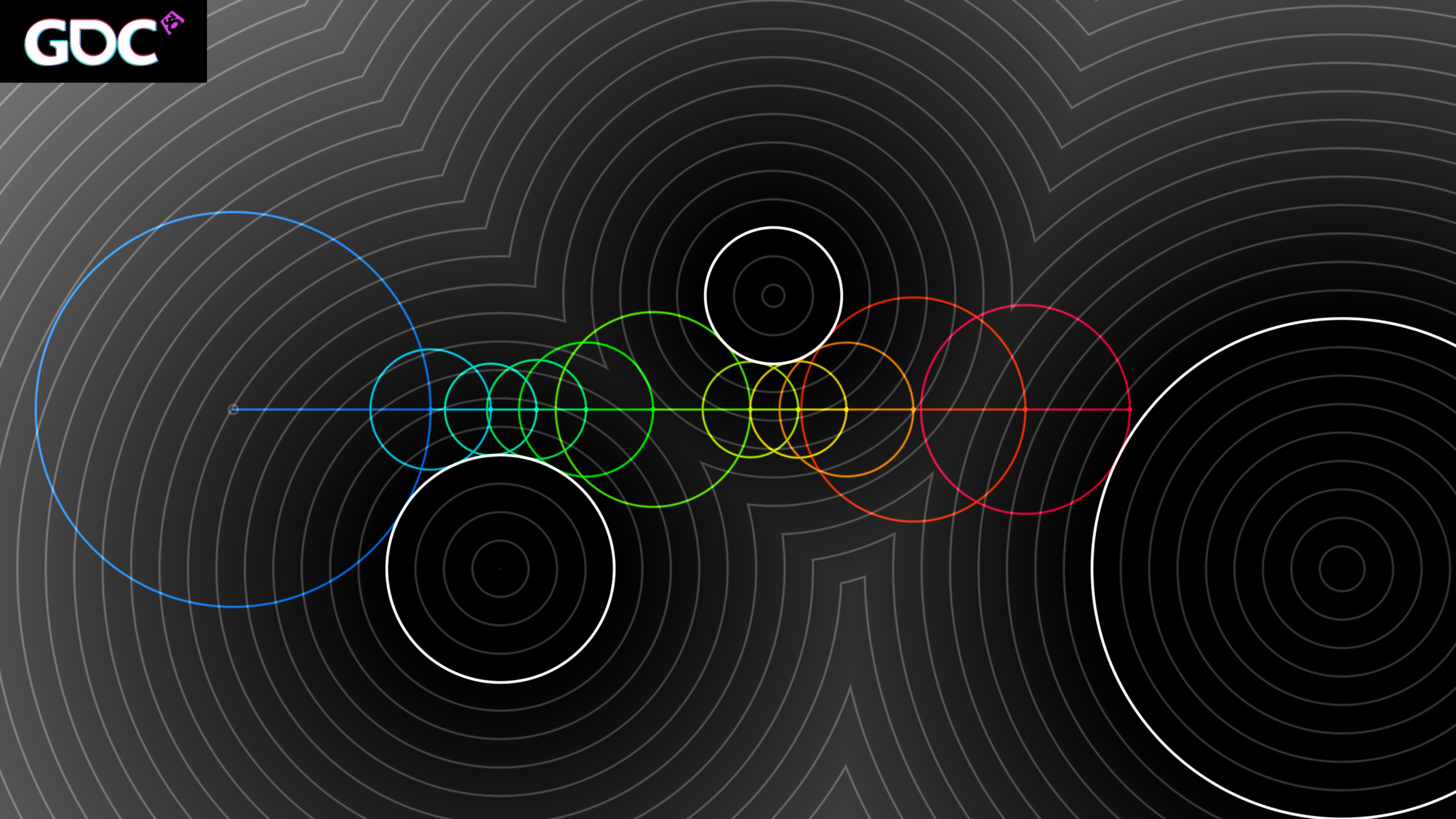


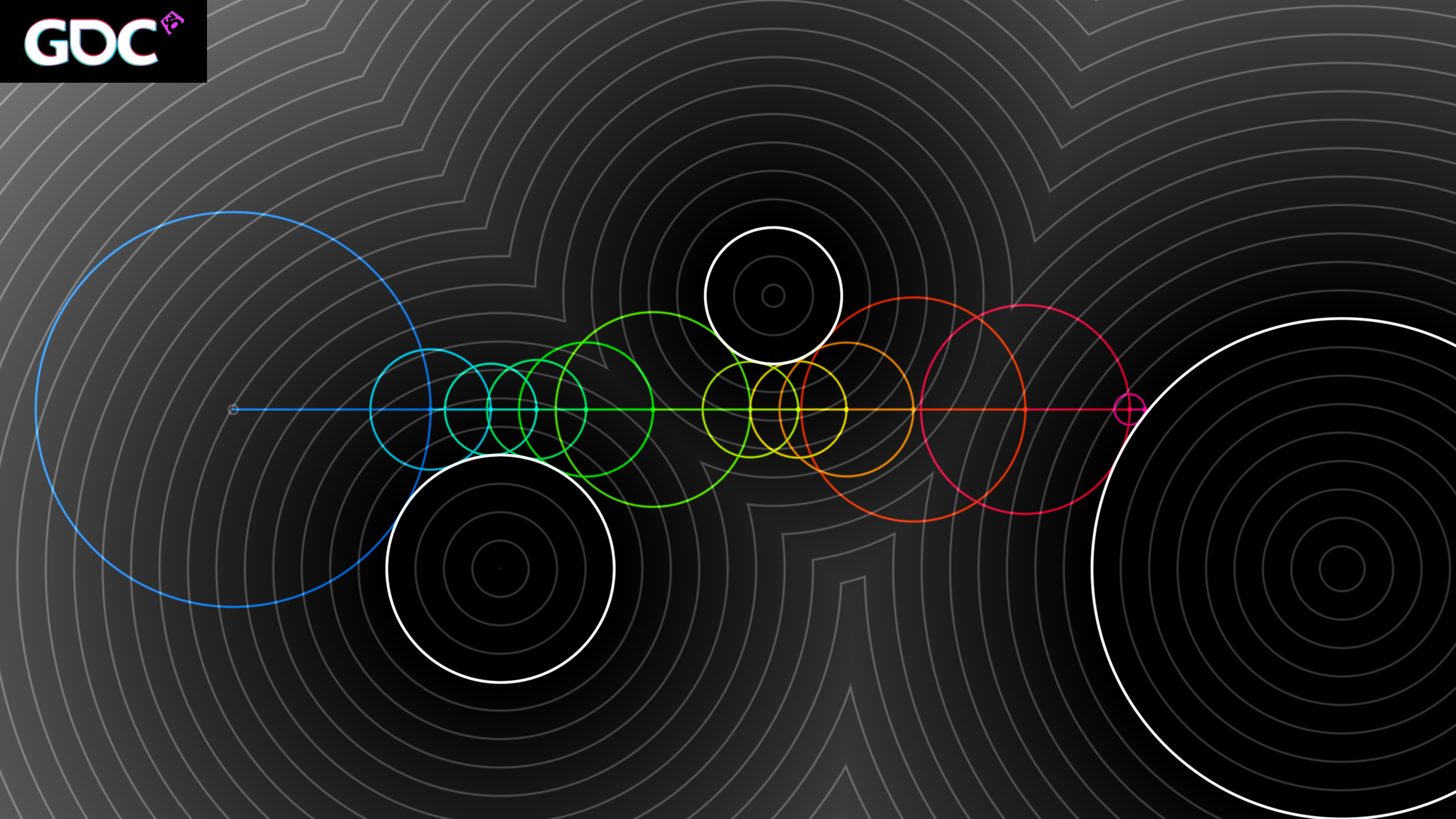


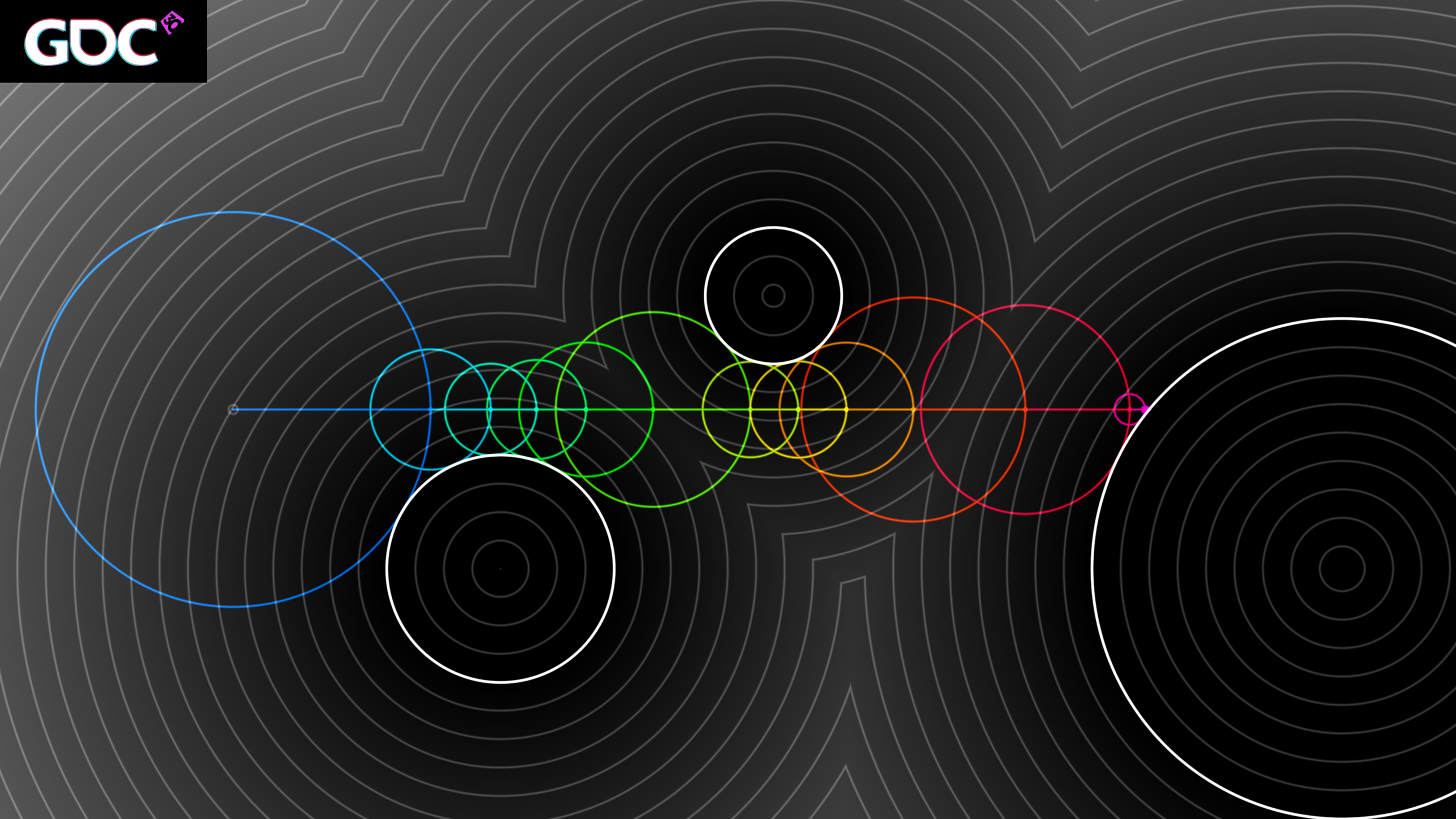


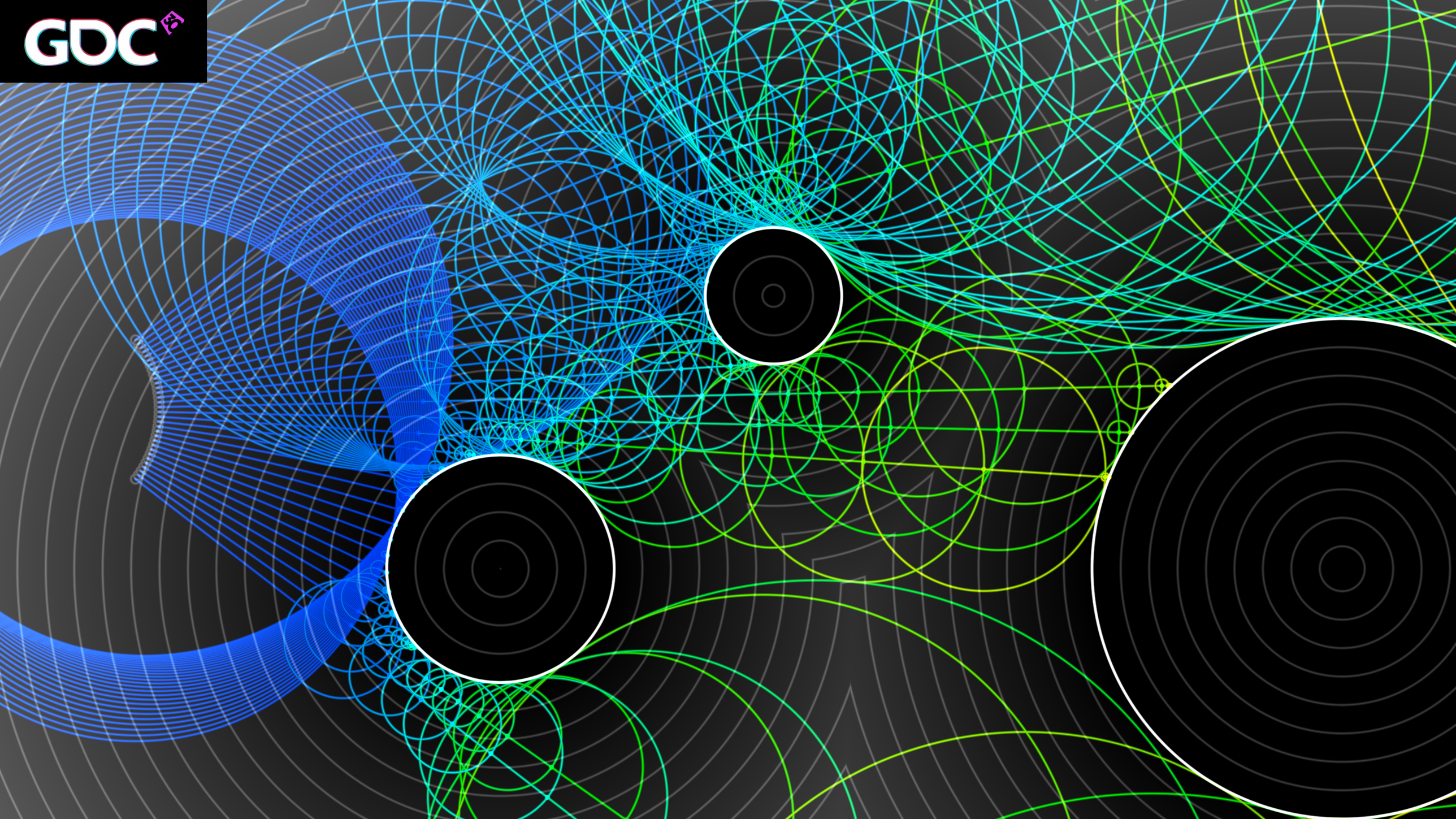


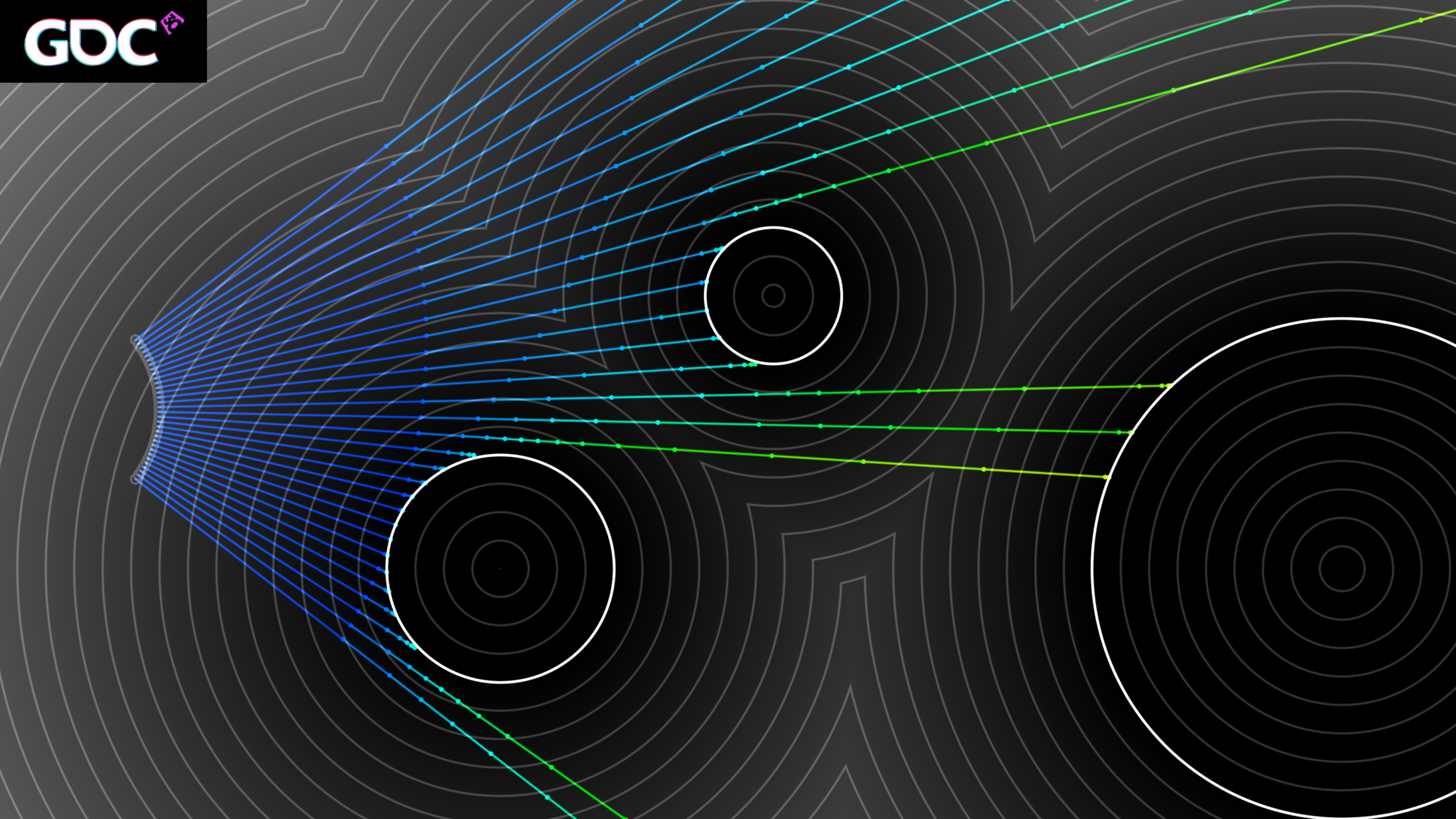


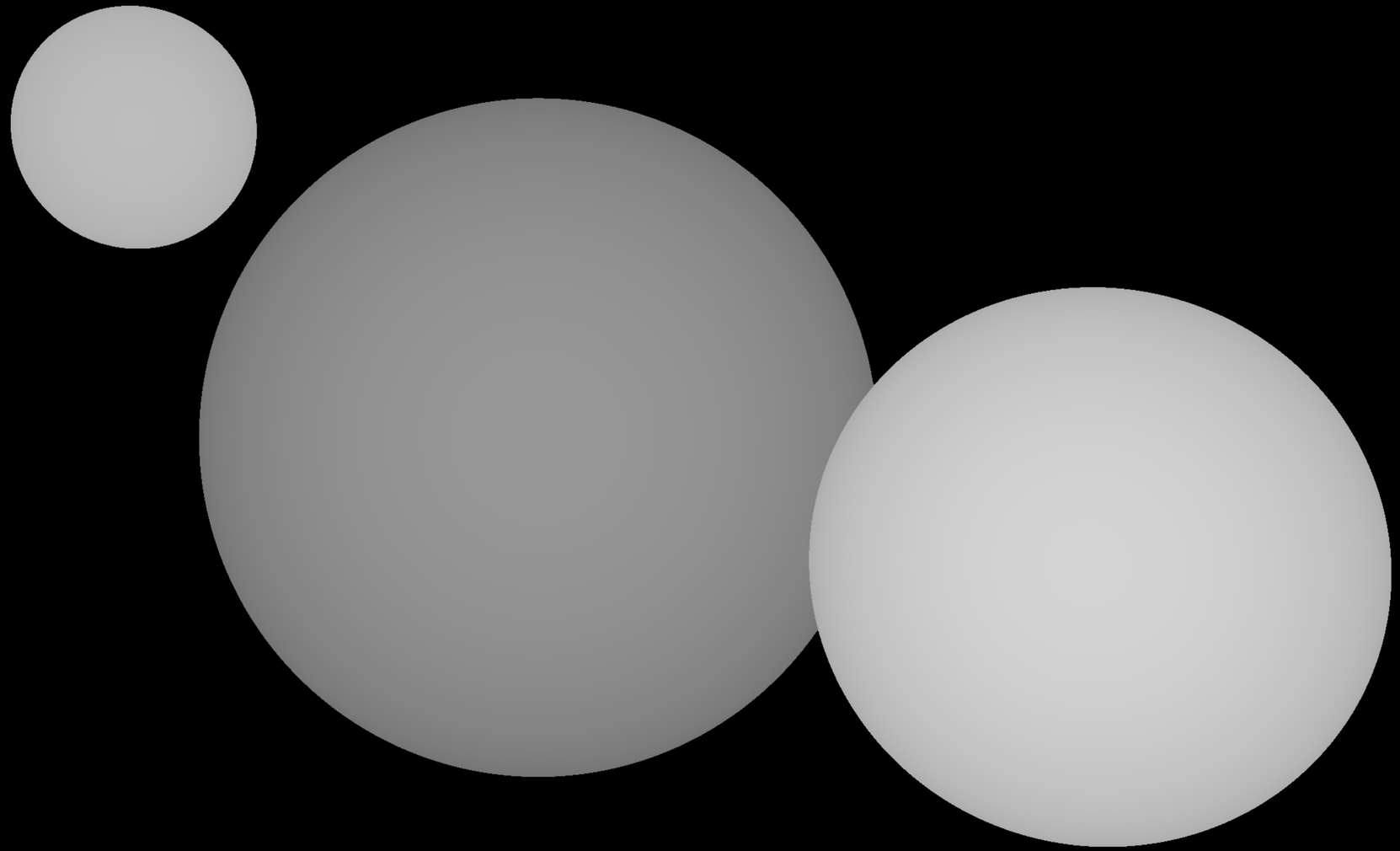


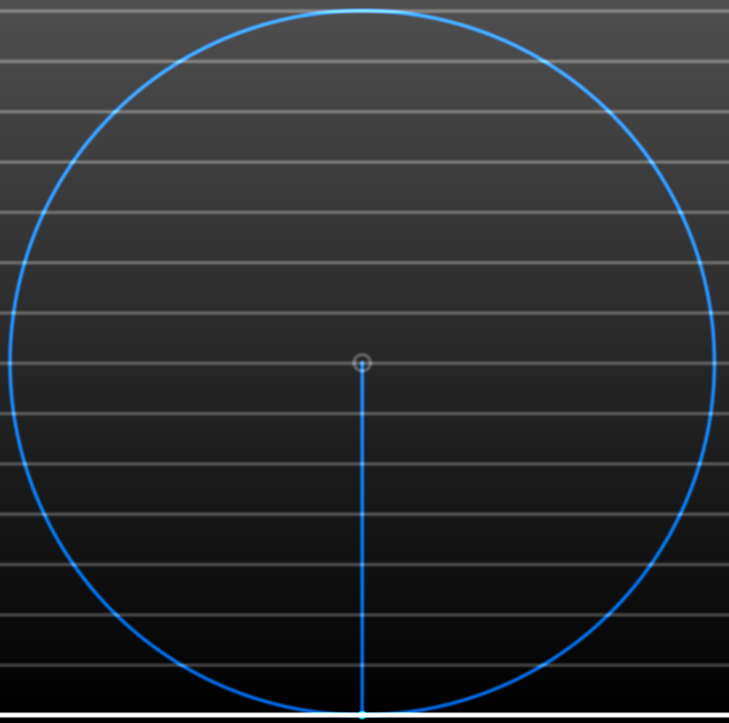


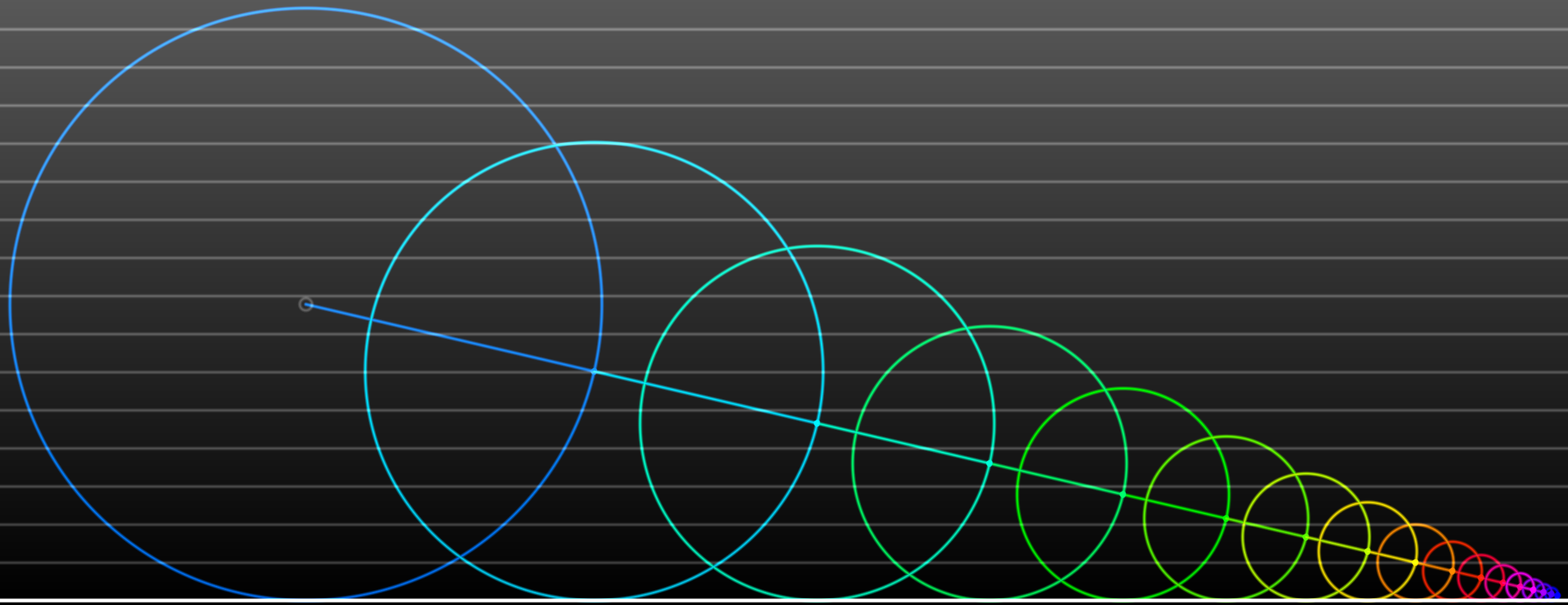

GDC

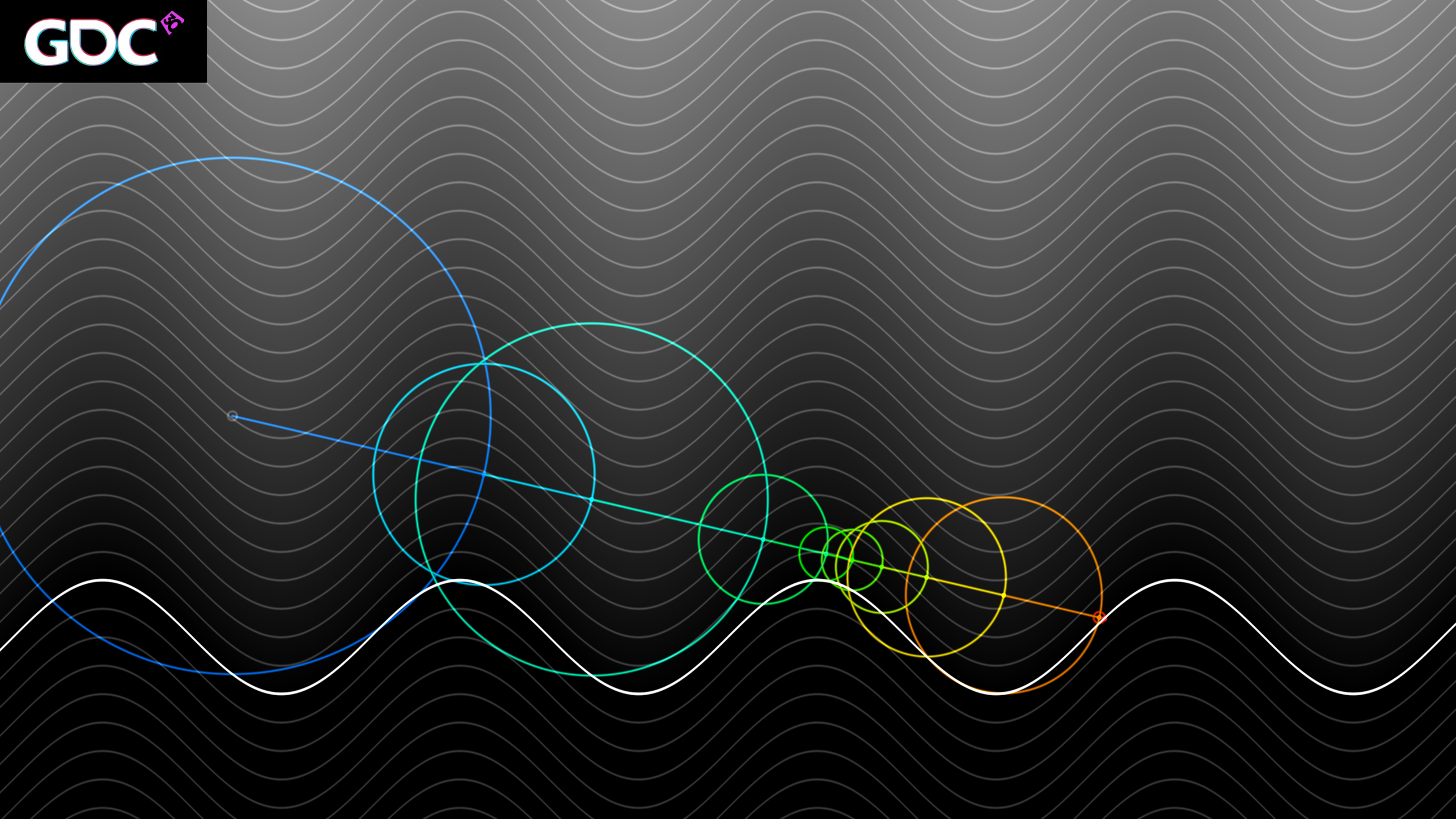

GDC

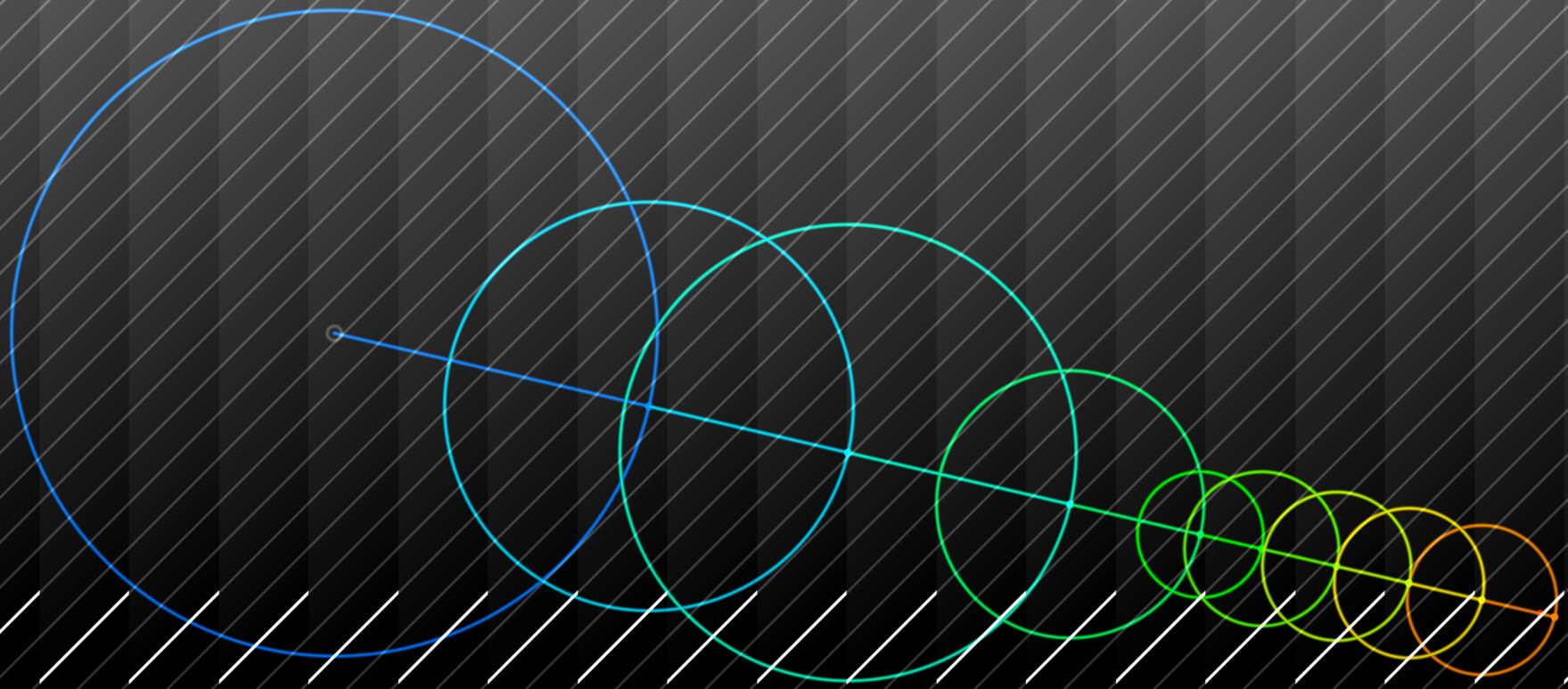

GDC

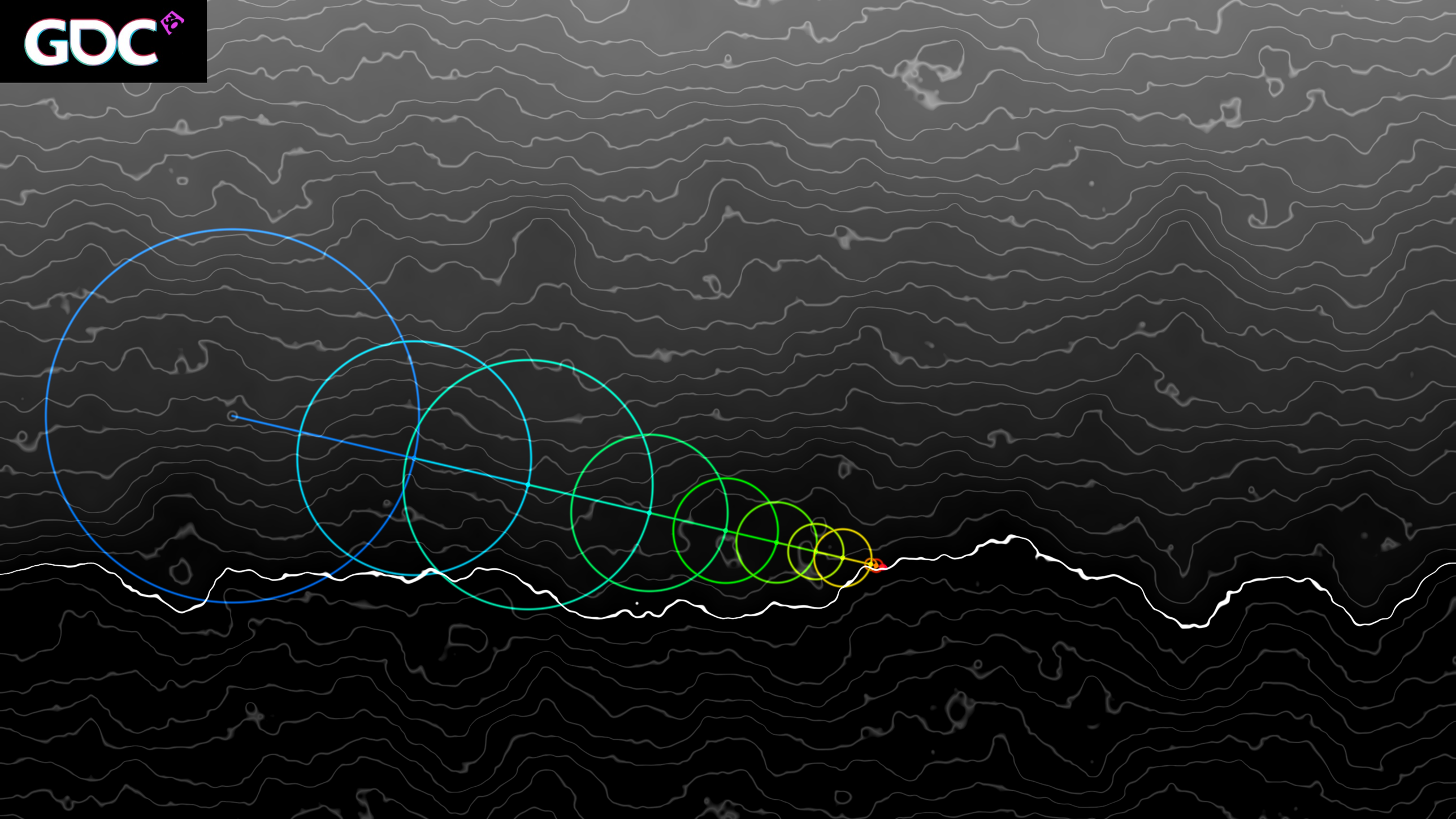


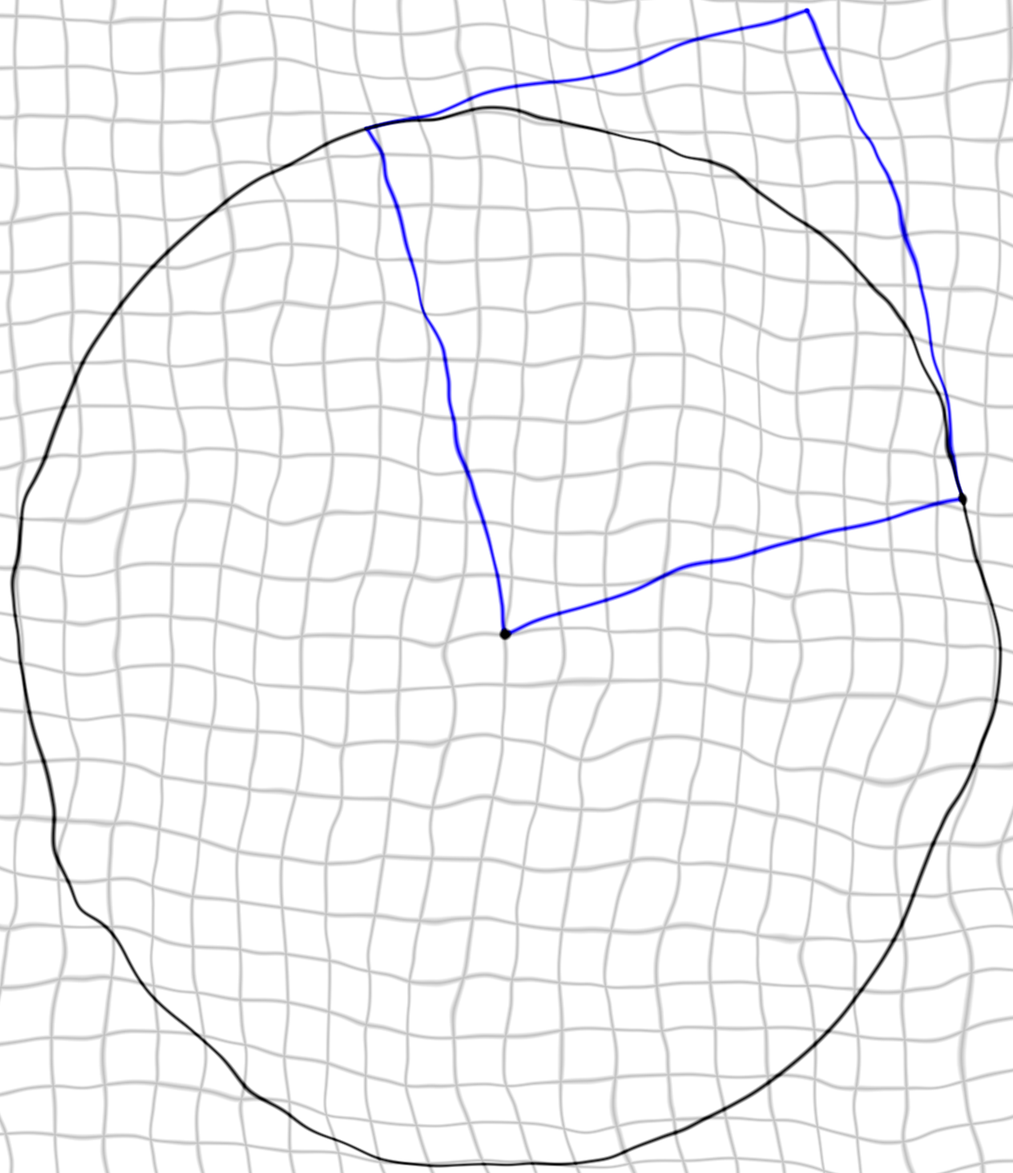


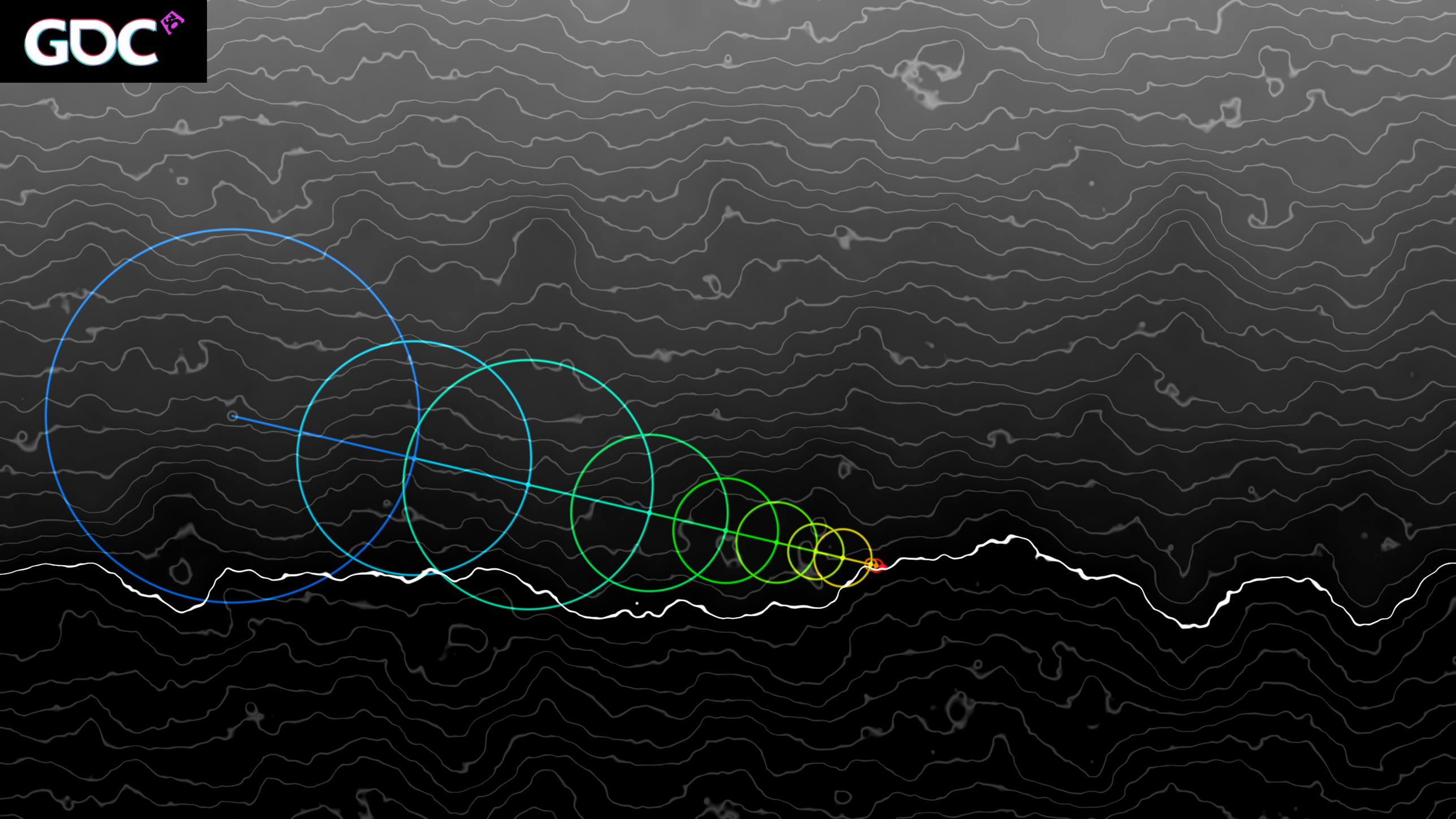


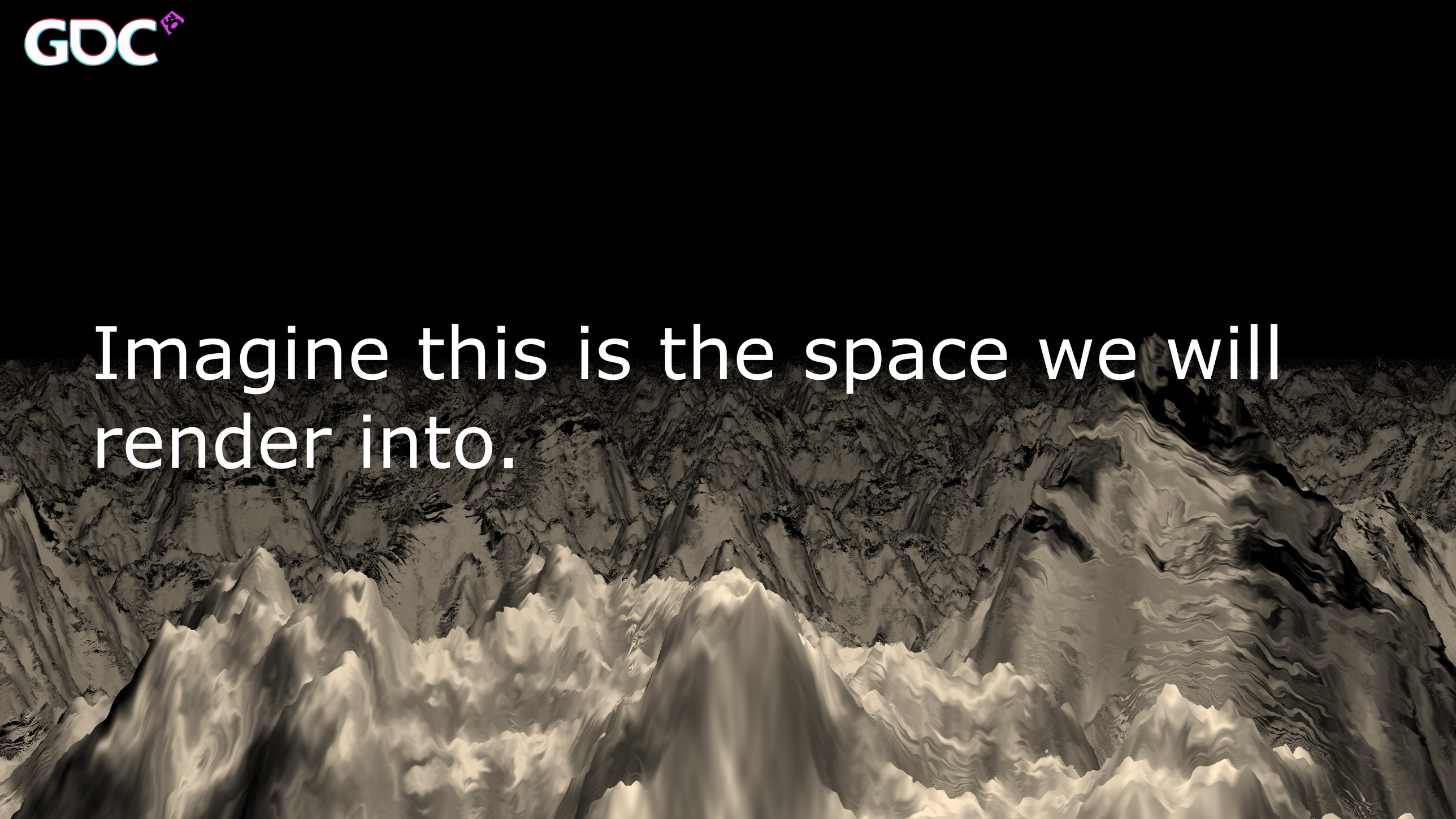


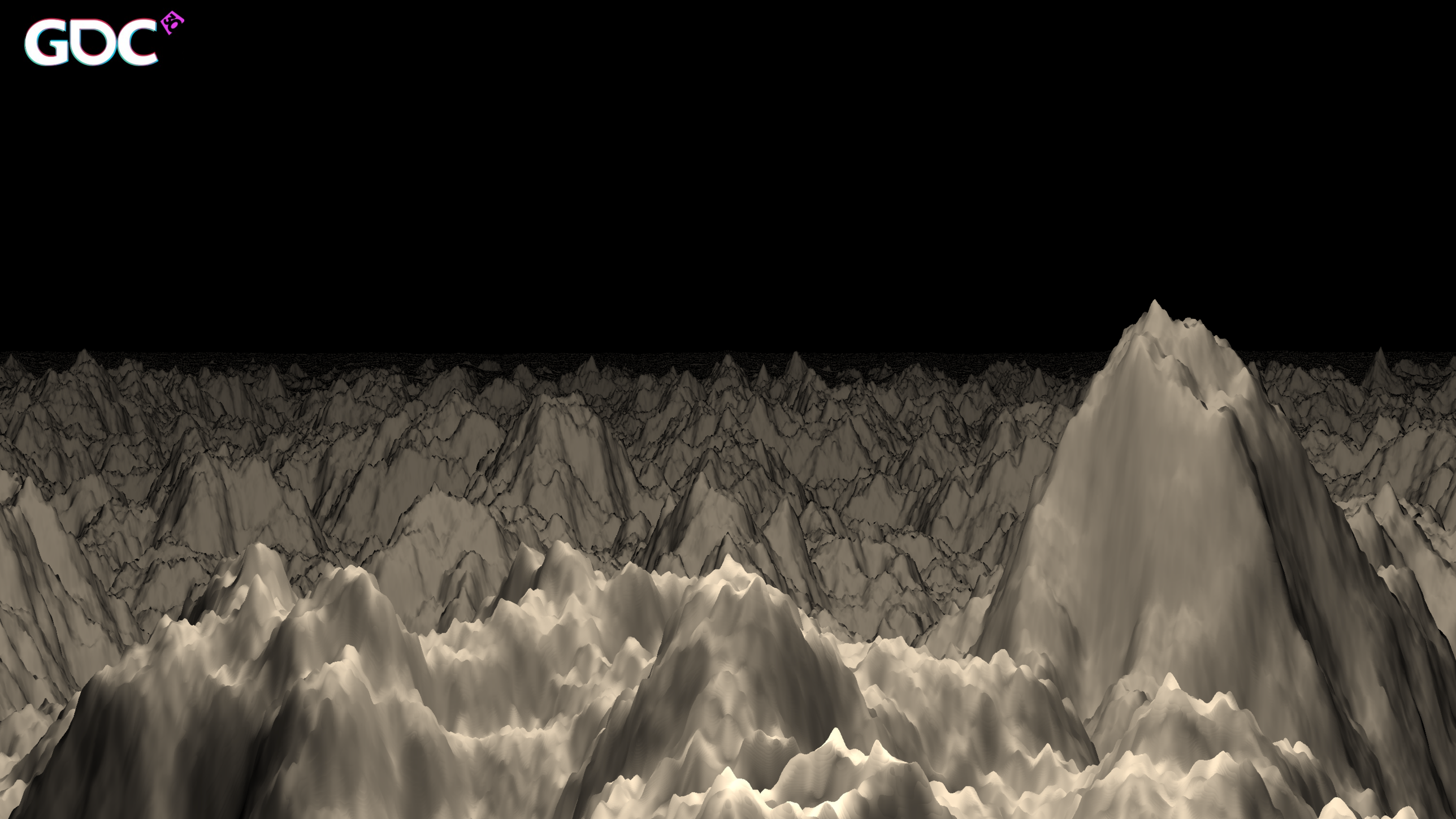


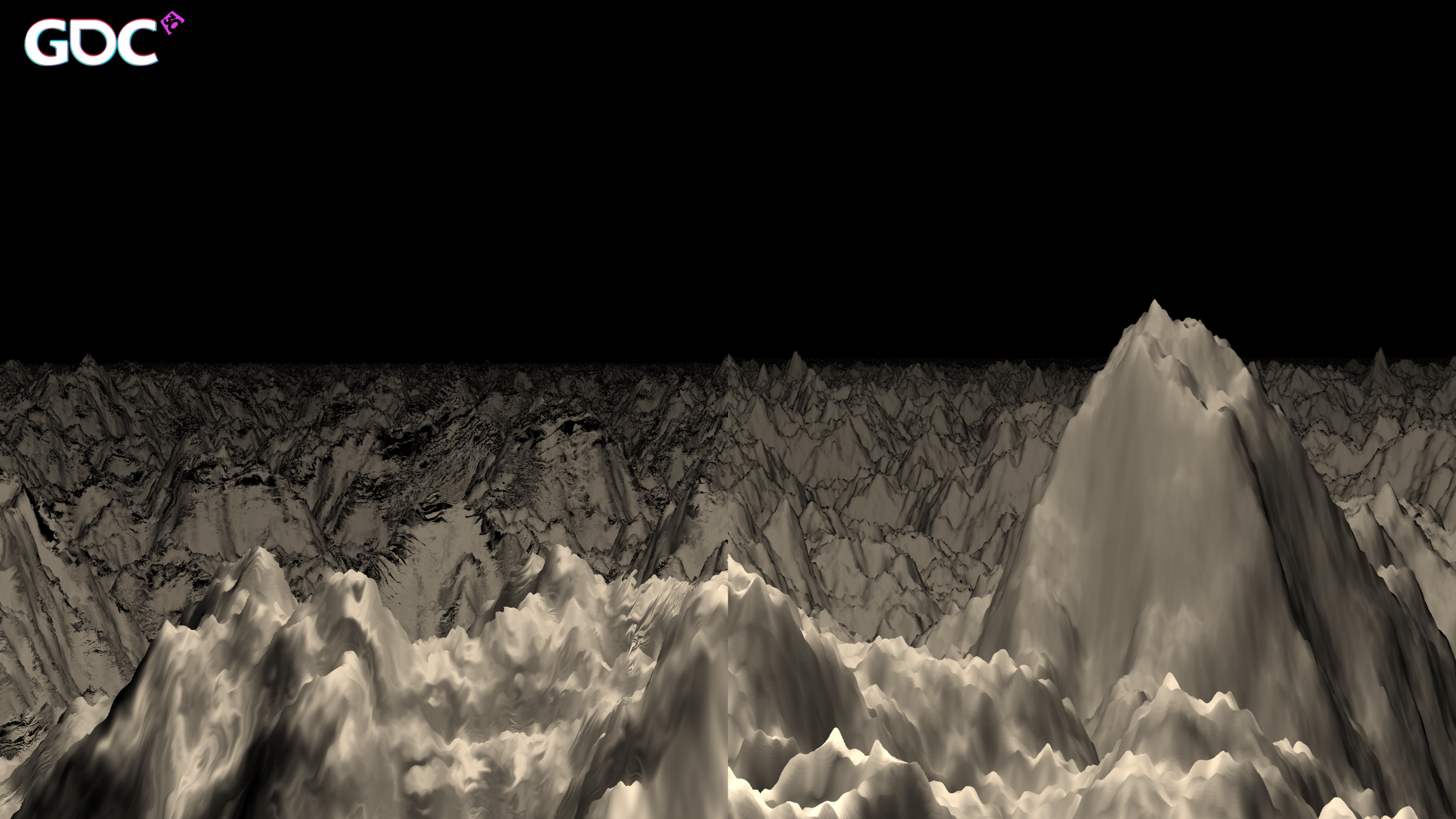


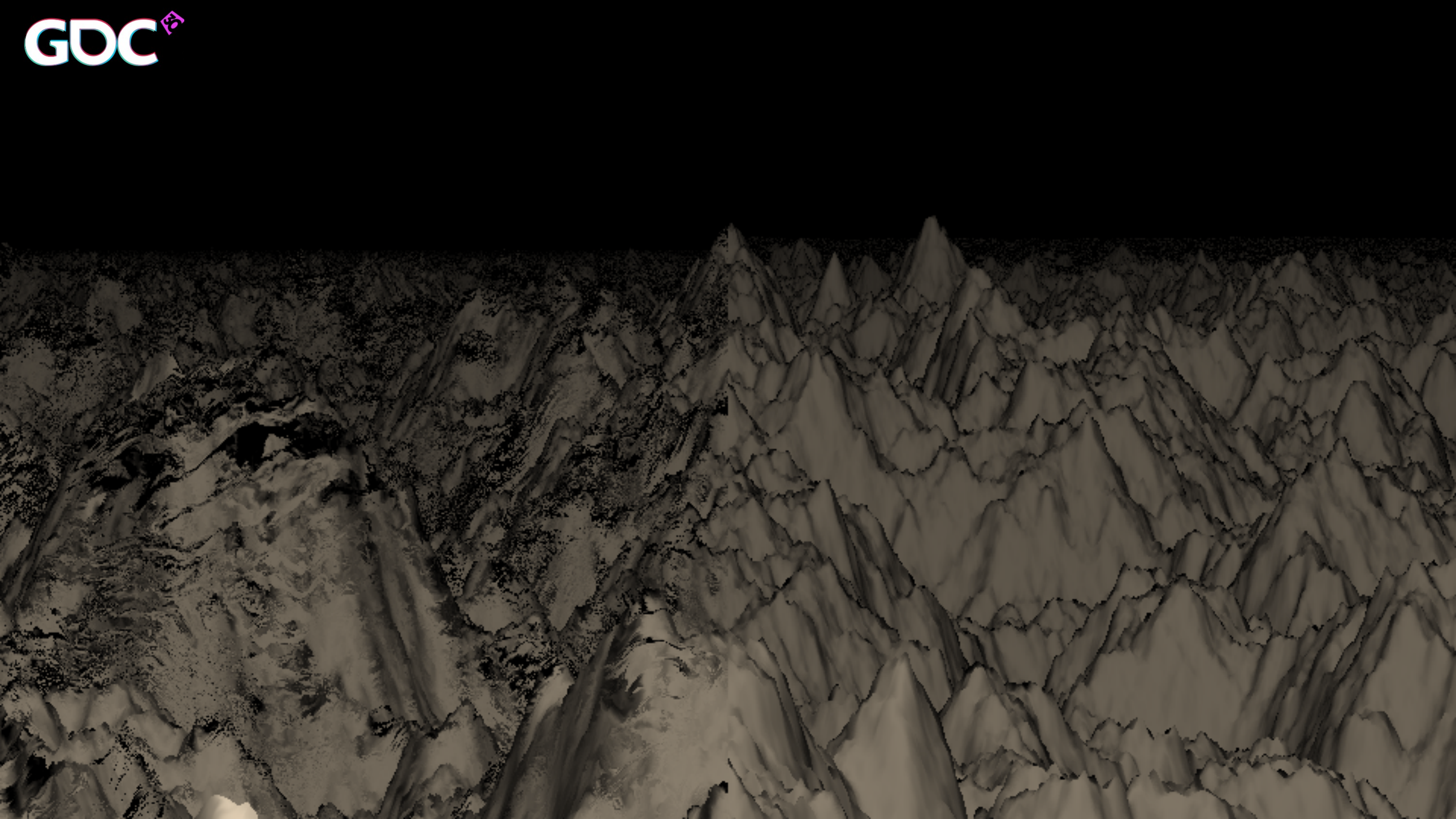




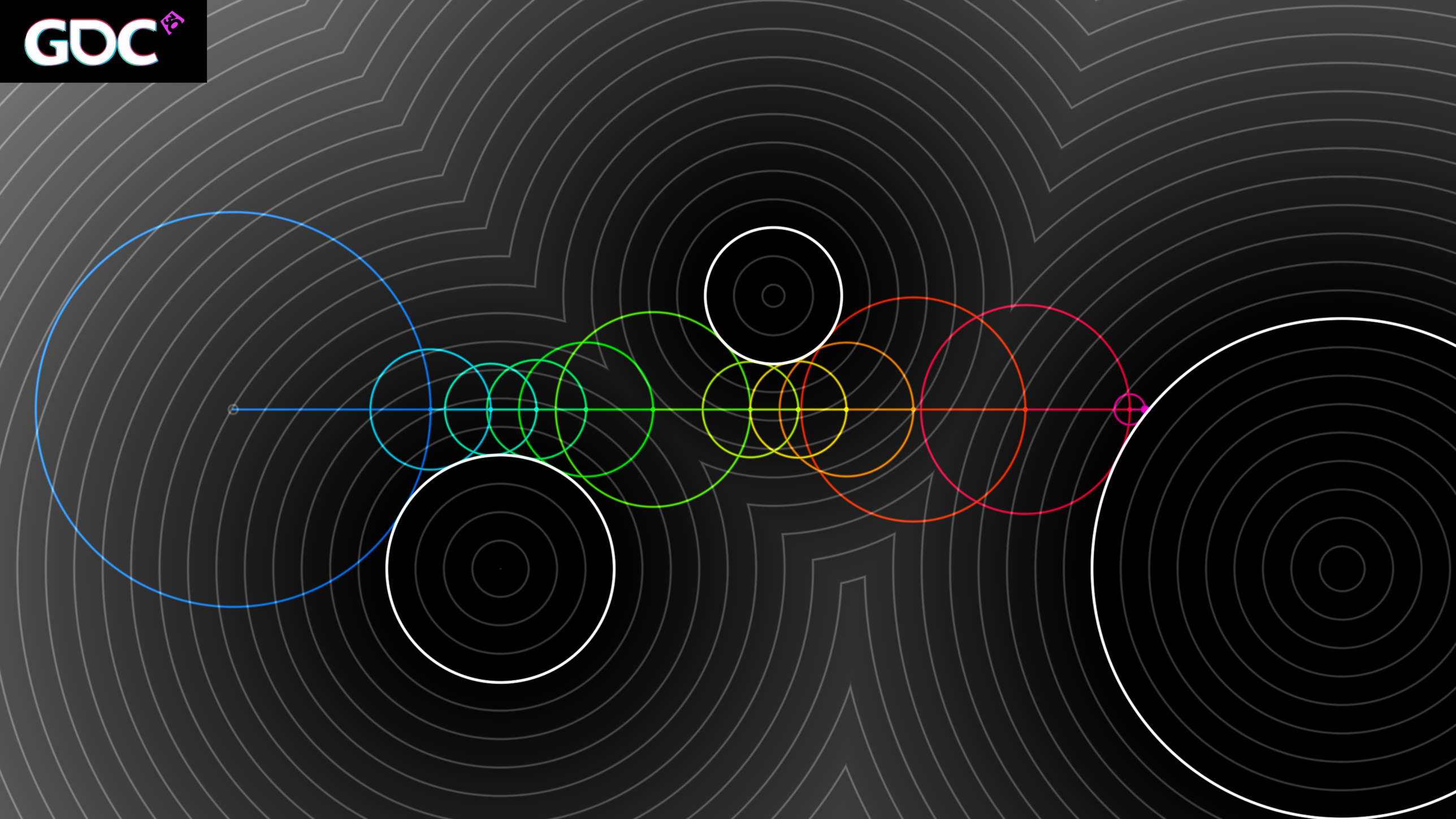

Imagine this is the space we will render into.

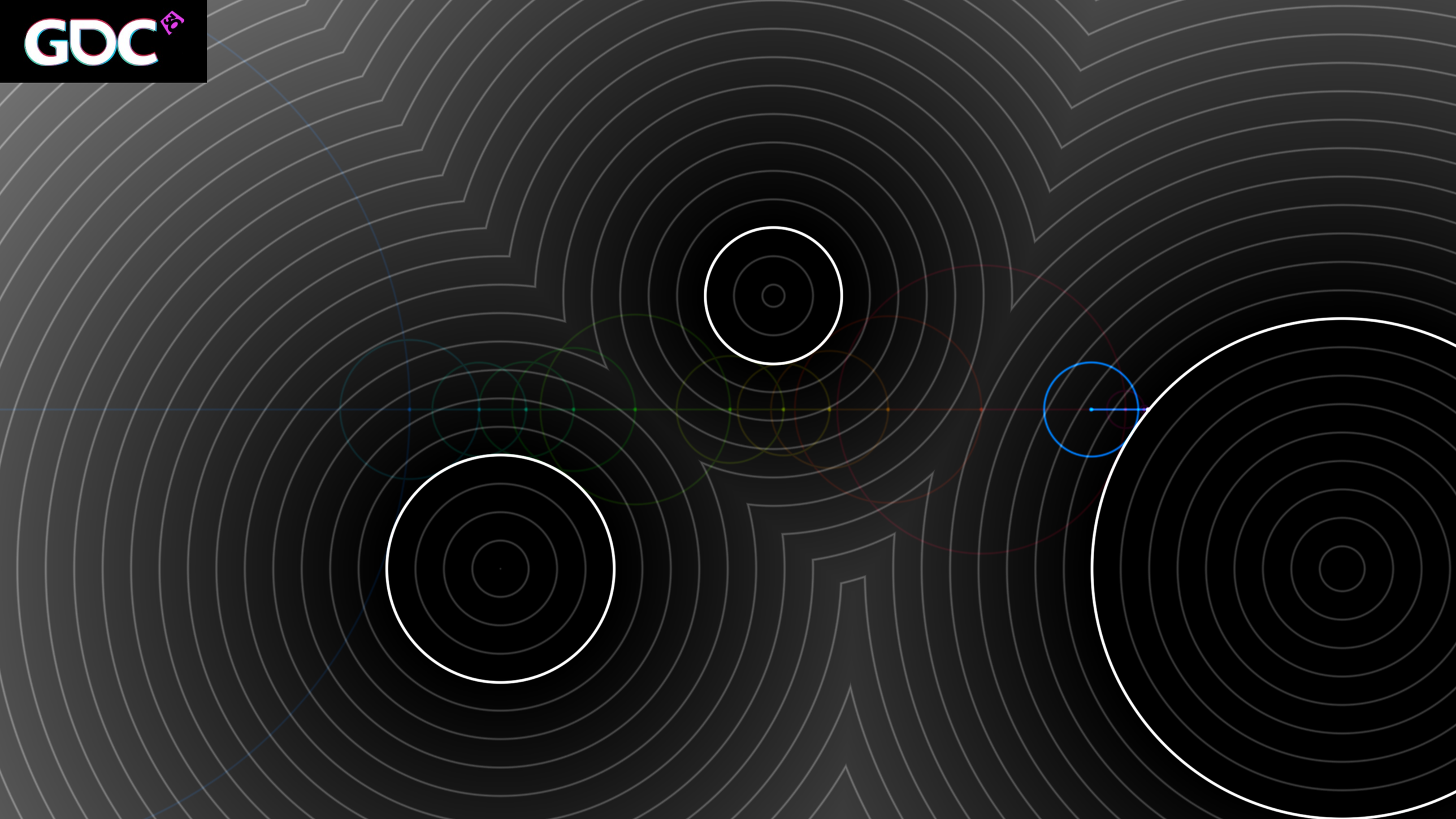


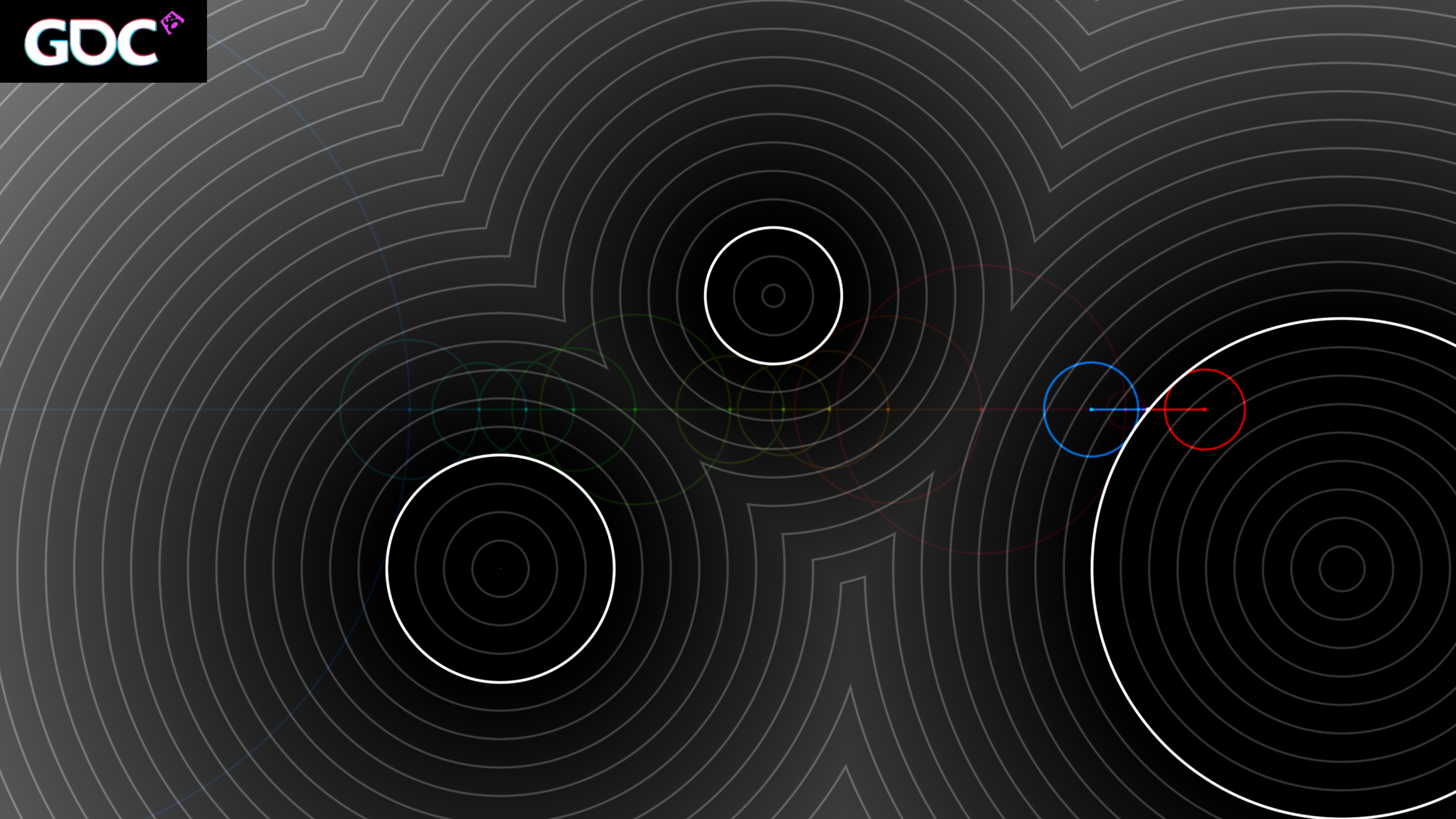


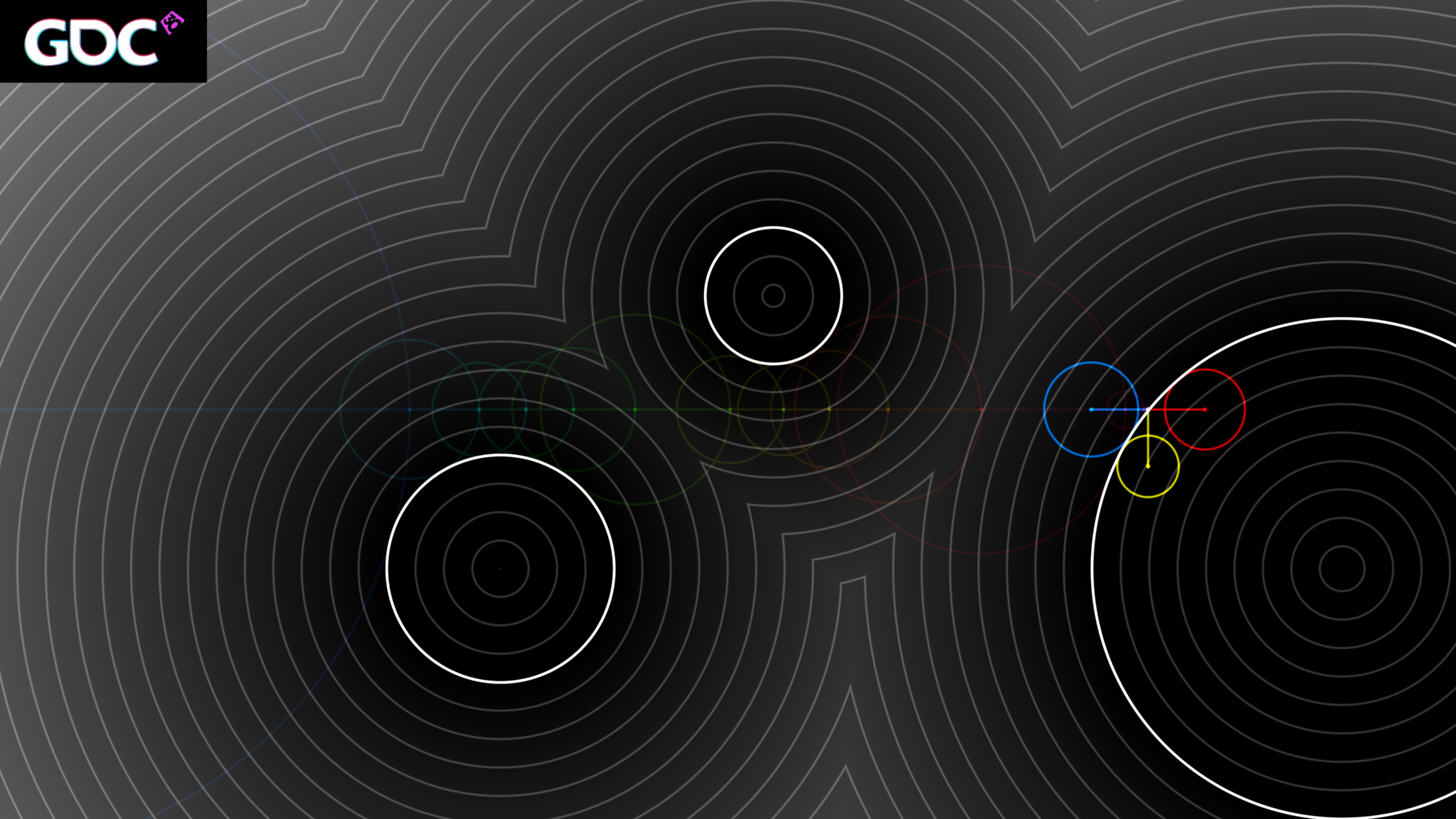


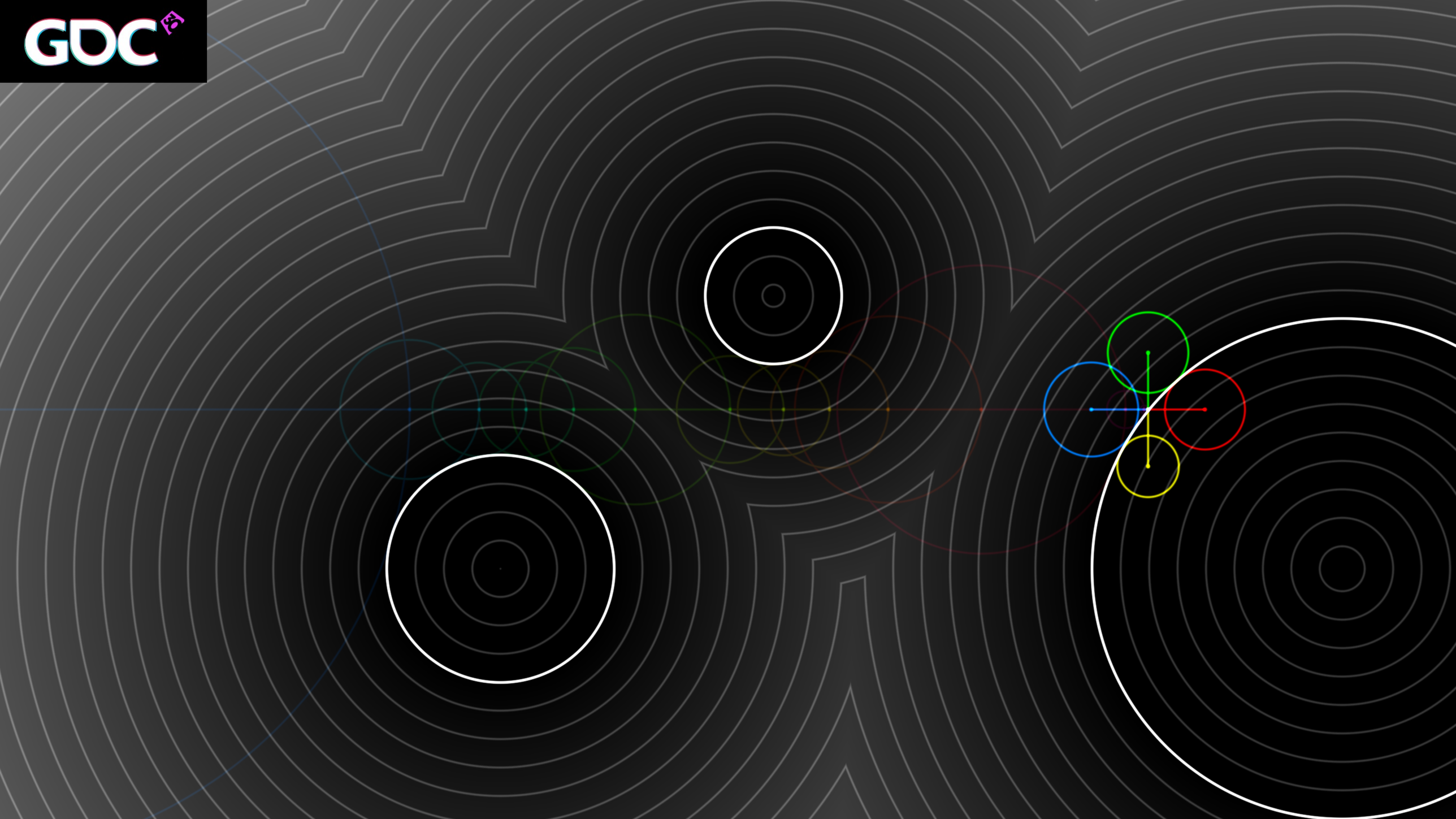
# Normal

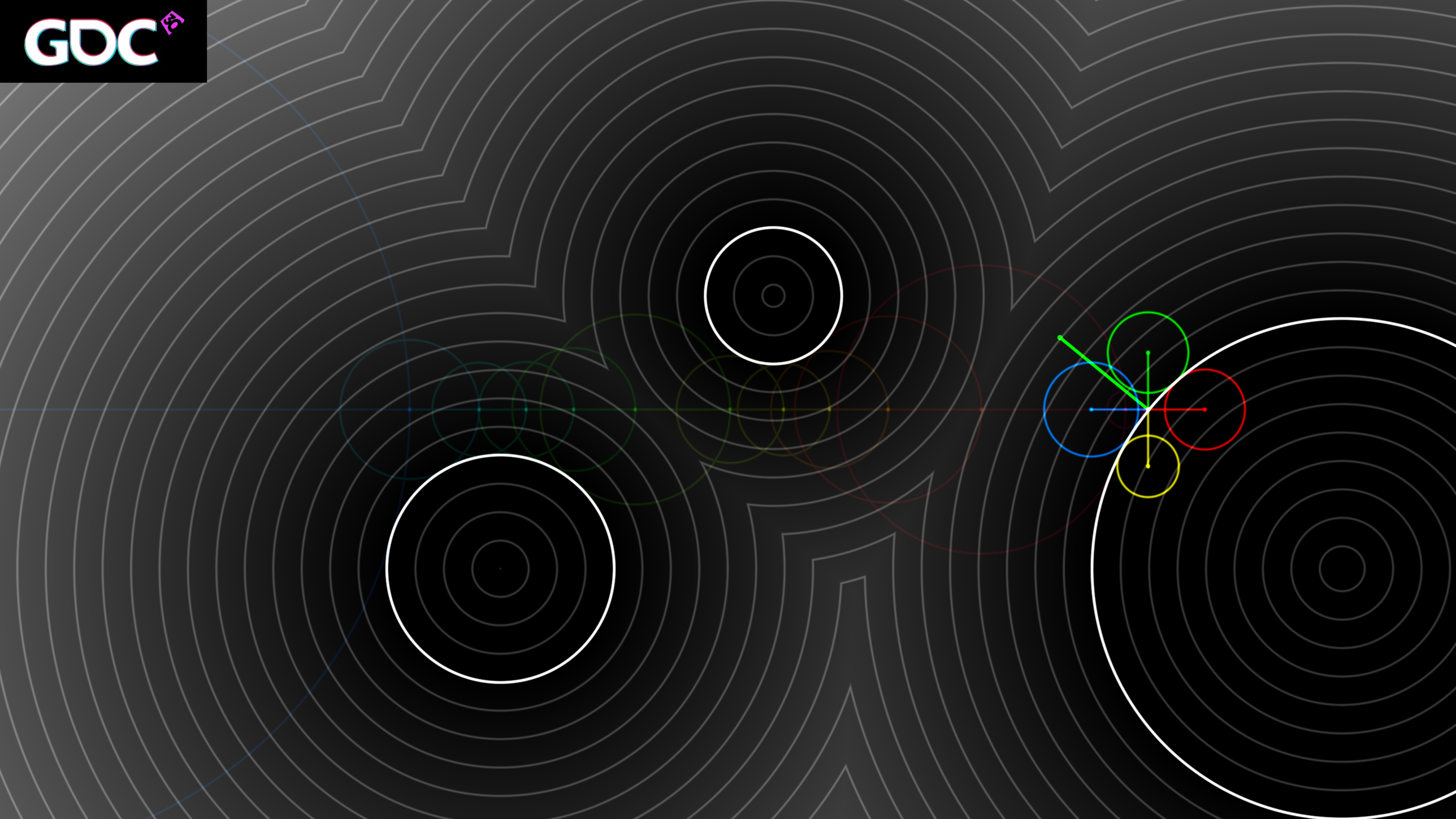

GDC

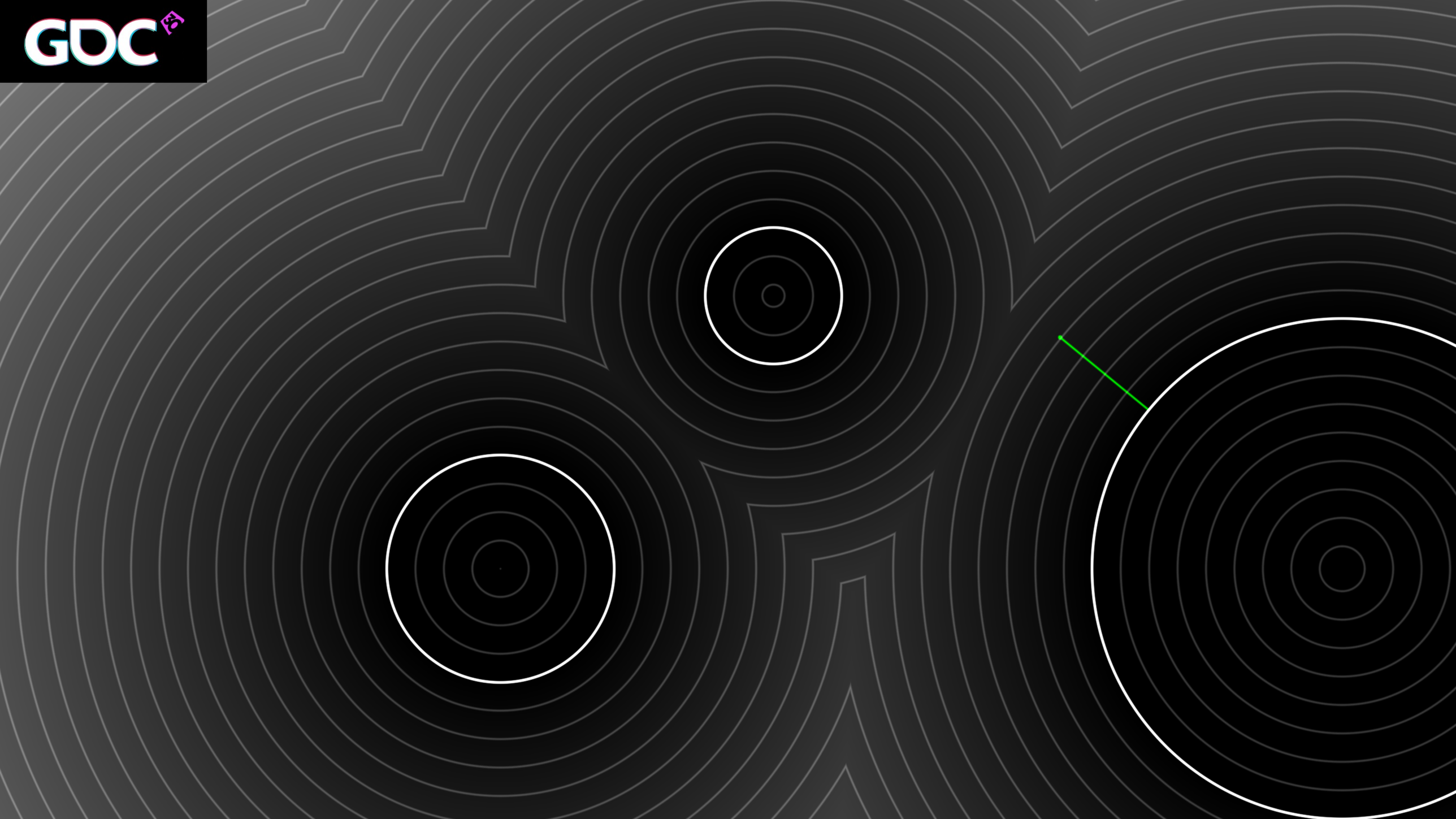


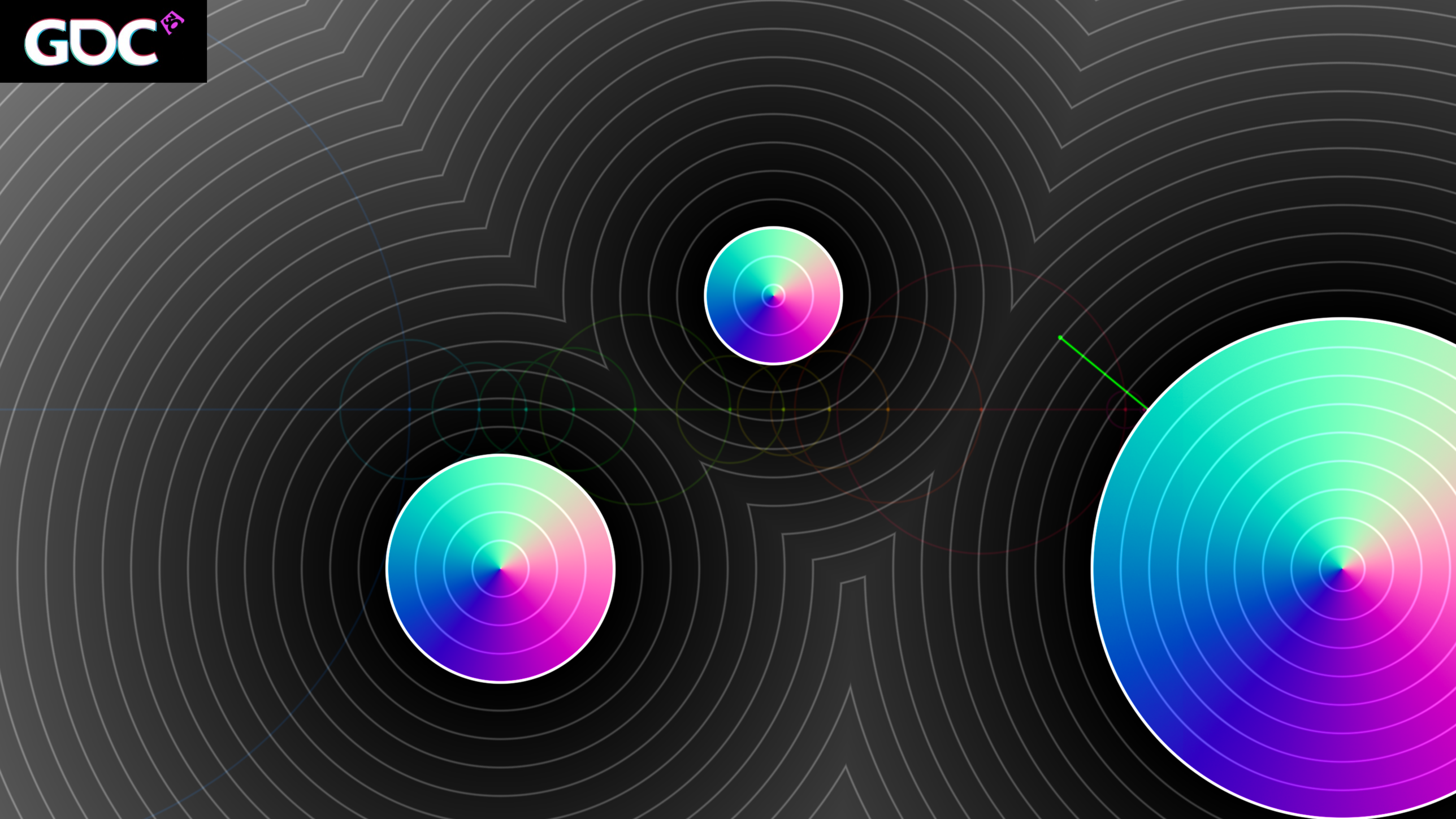

GDC

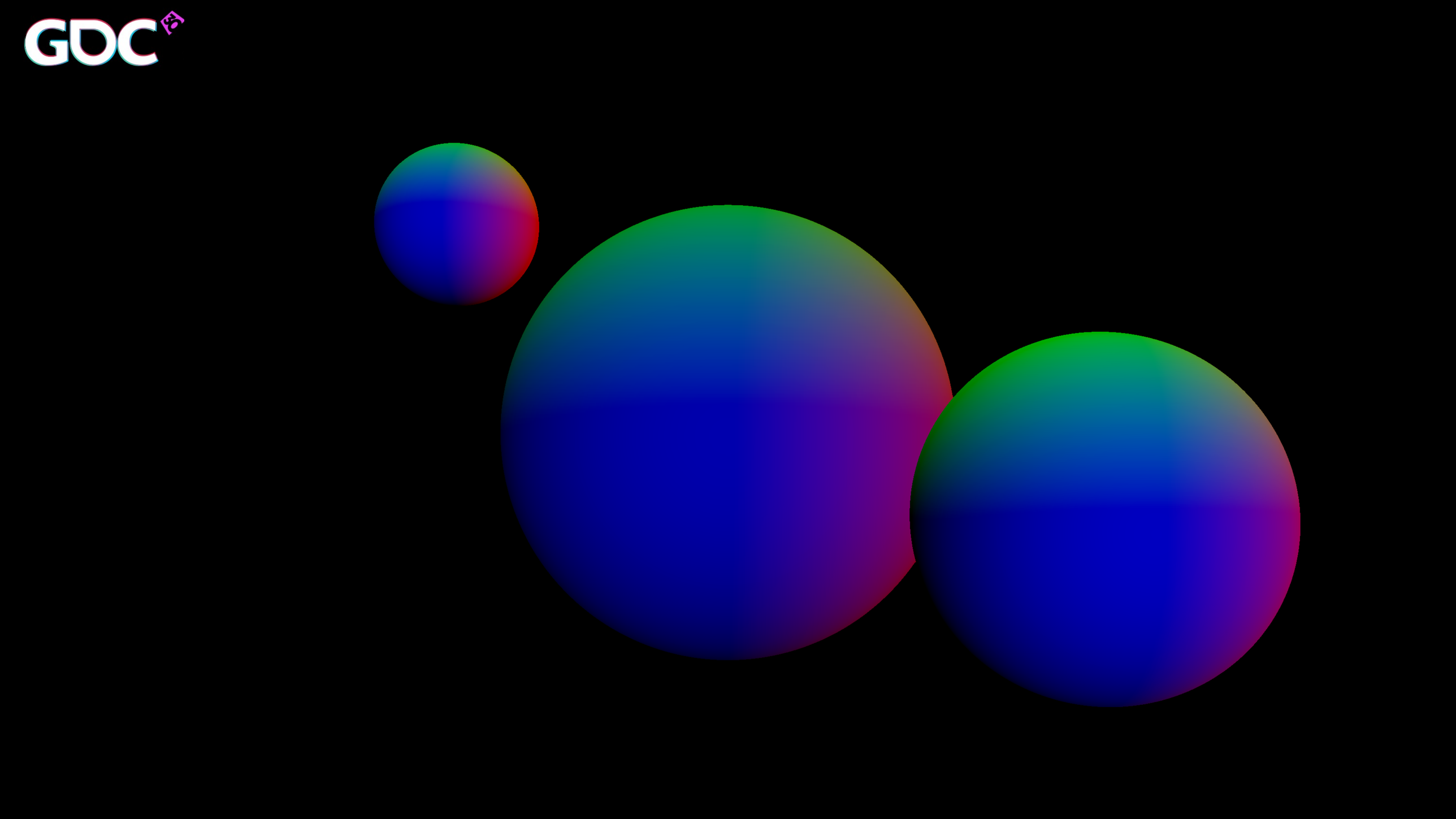

GDC

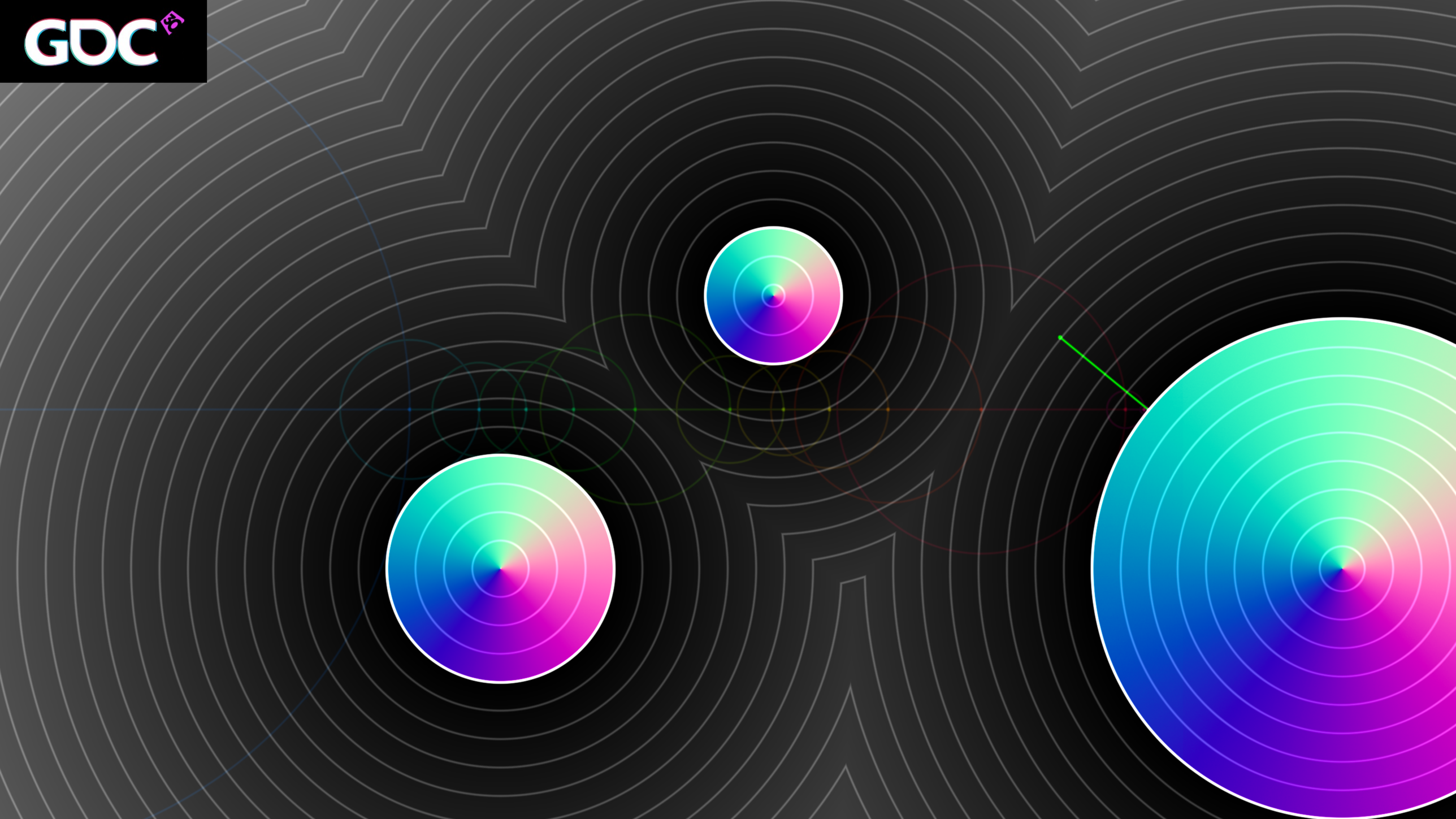

GDC

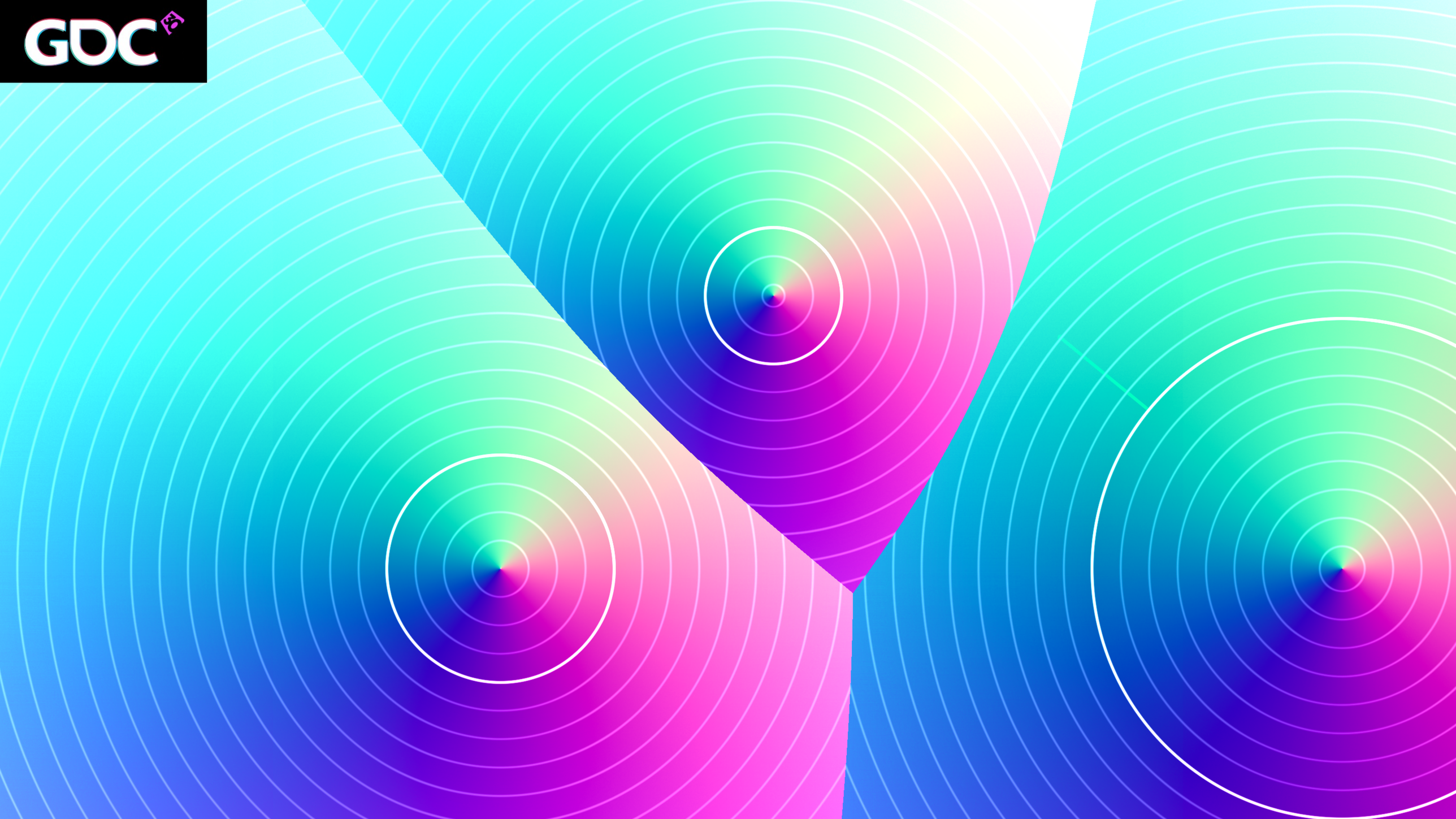


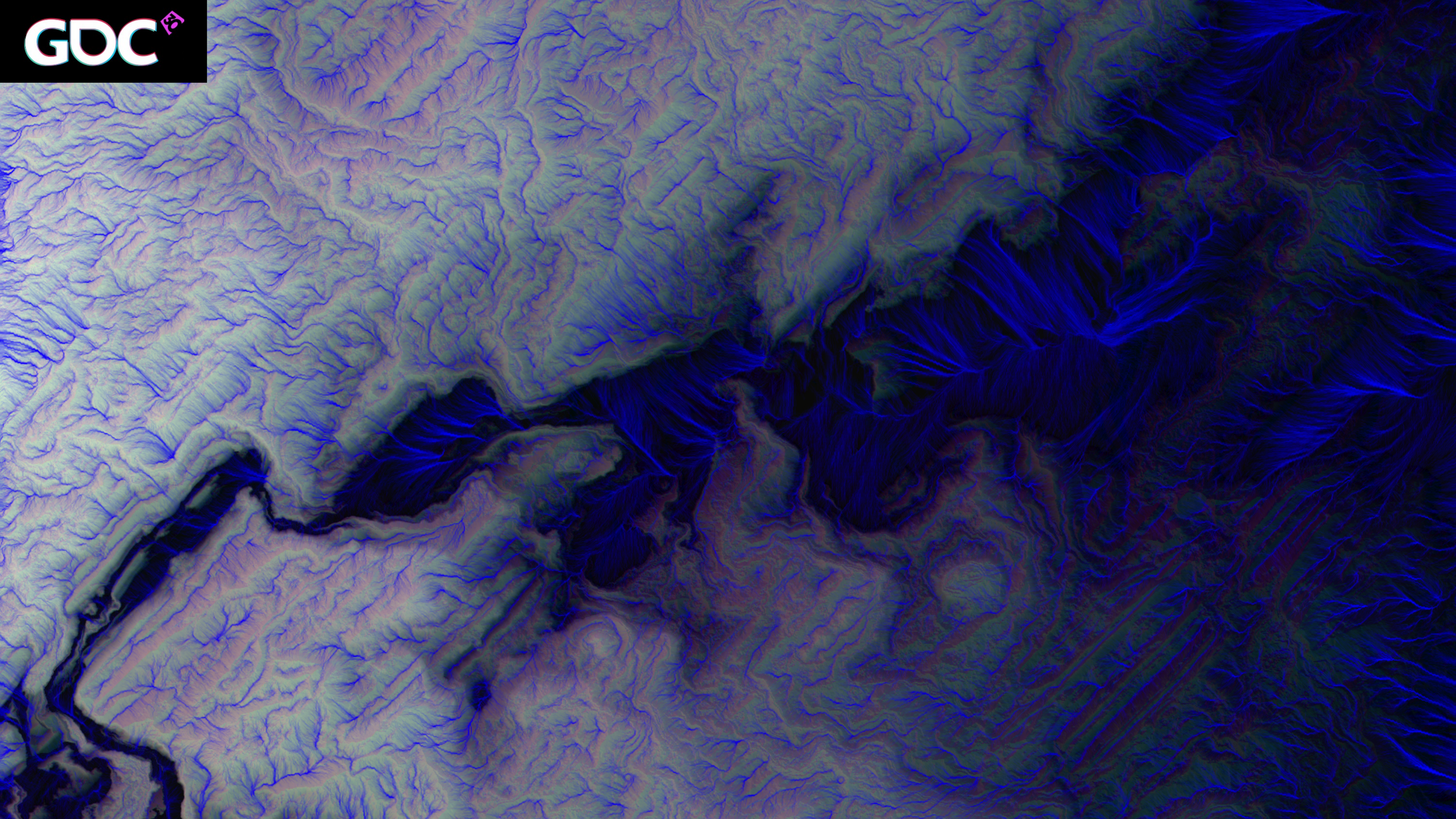


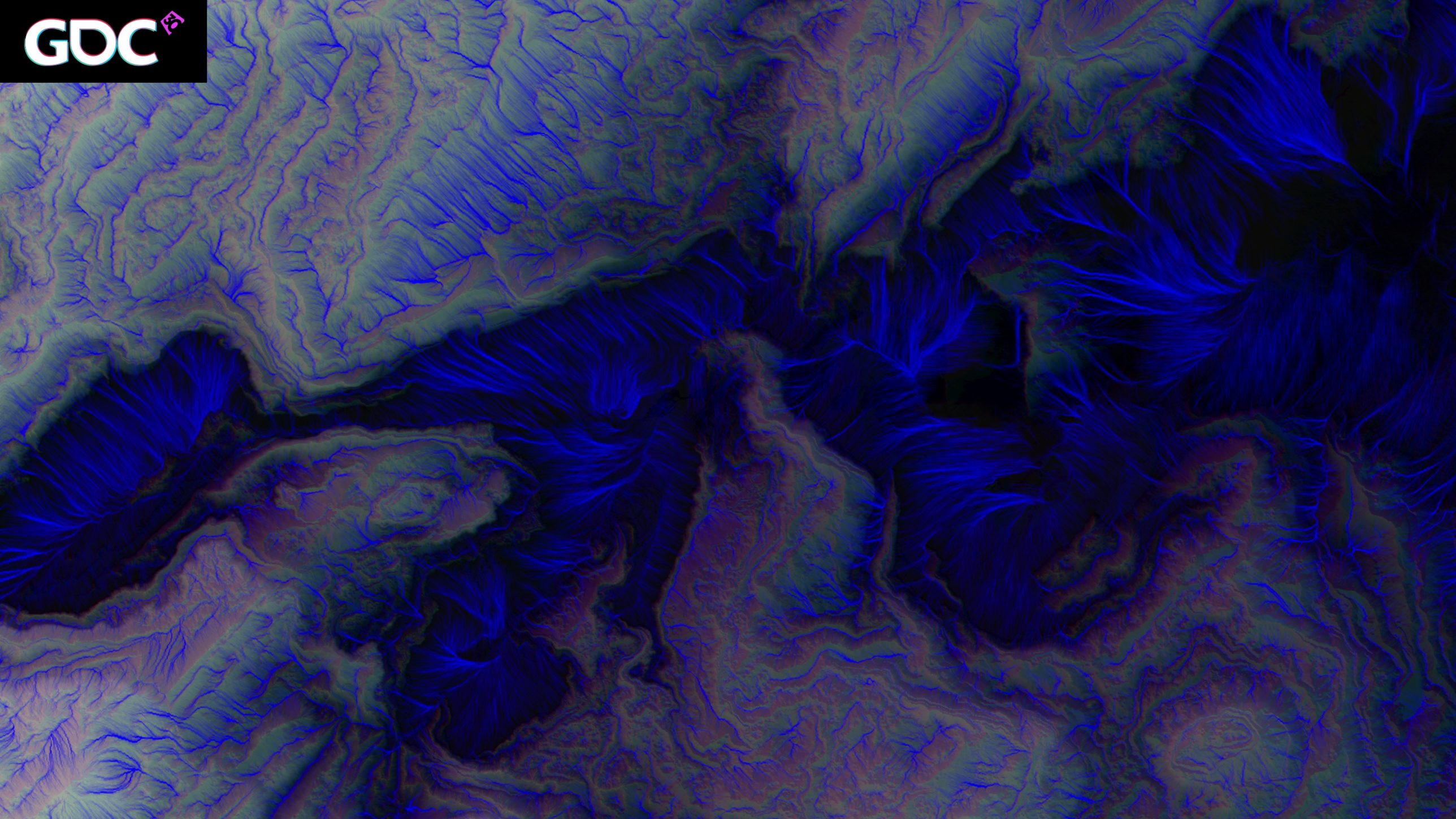

GDC

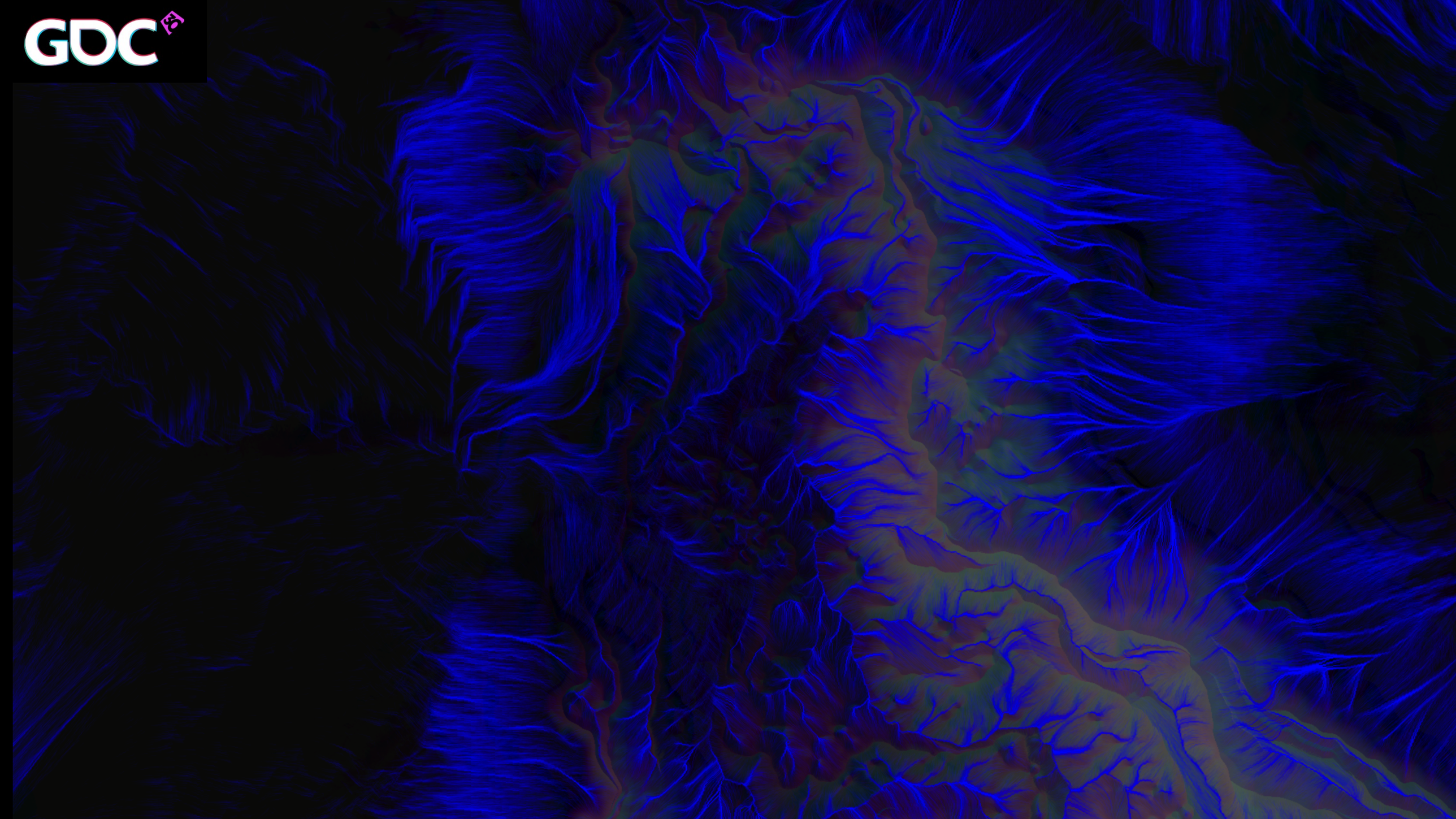


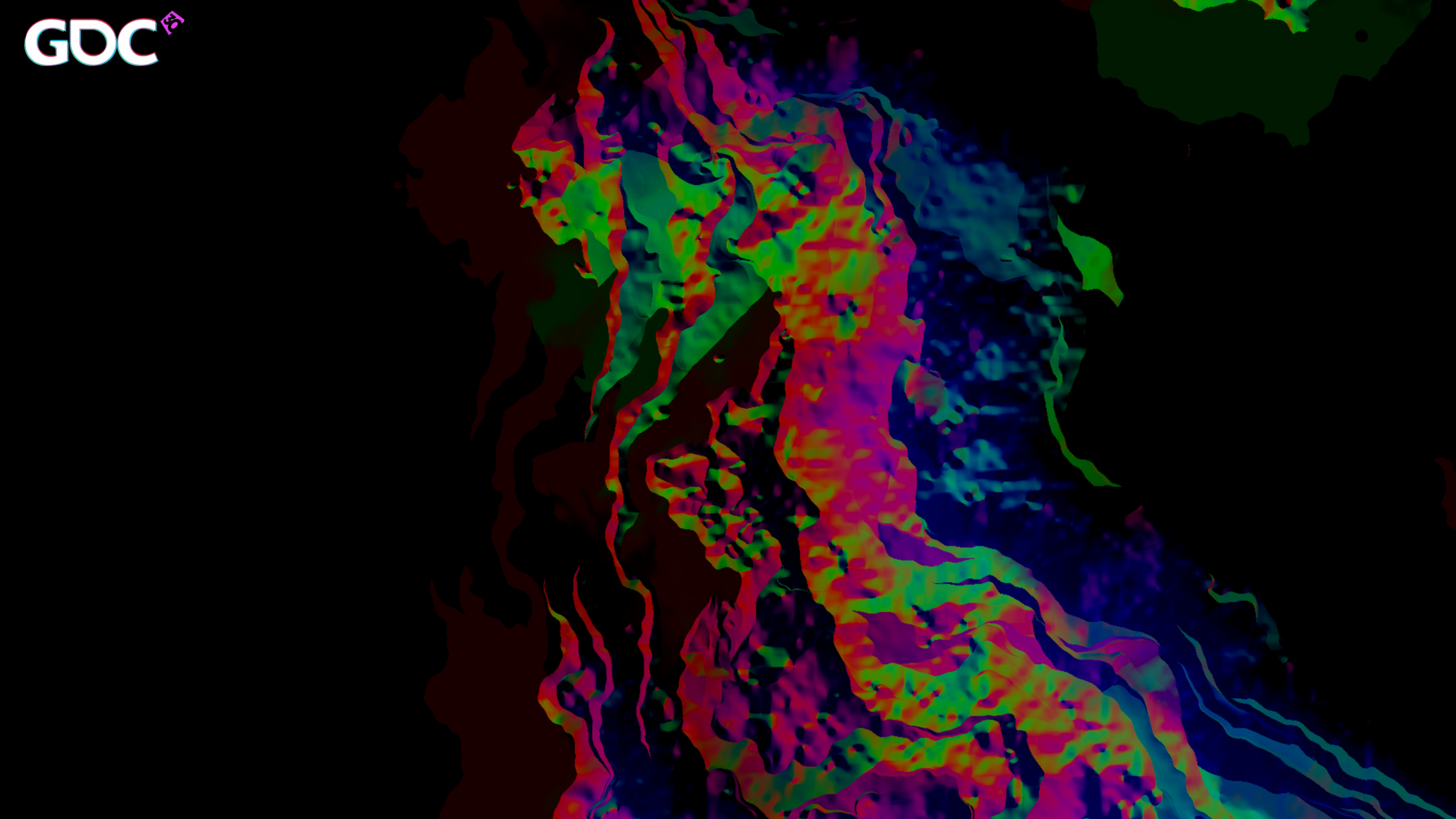

GDC


GDC

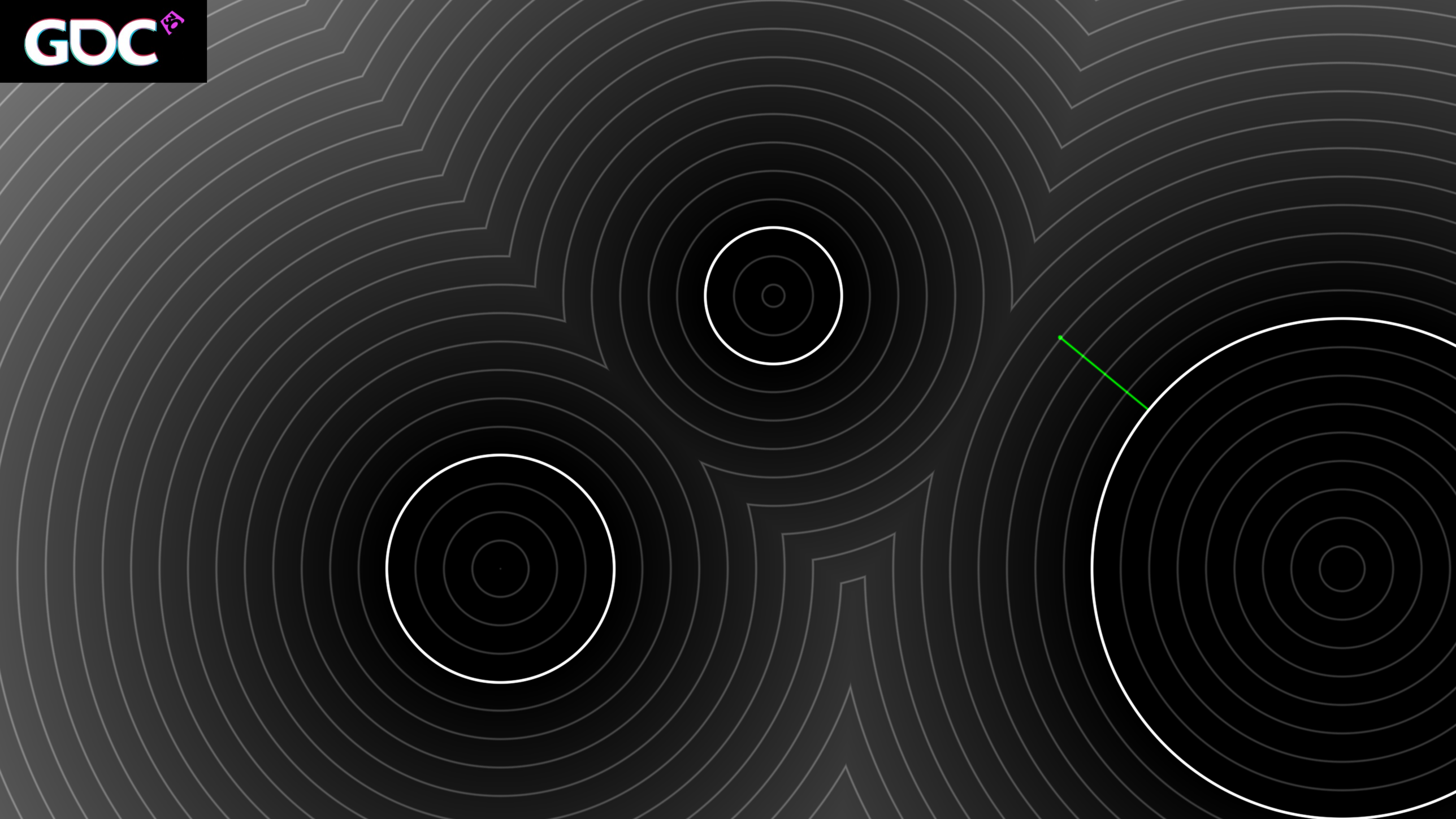

GDC

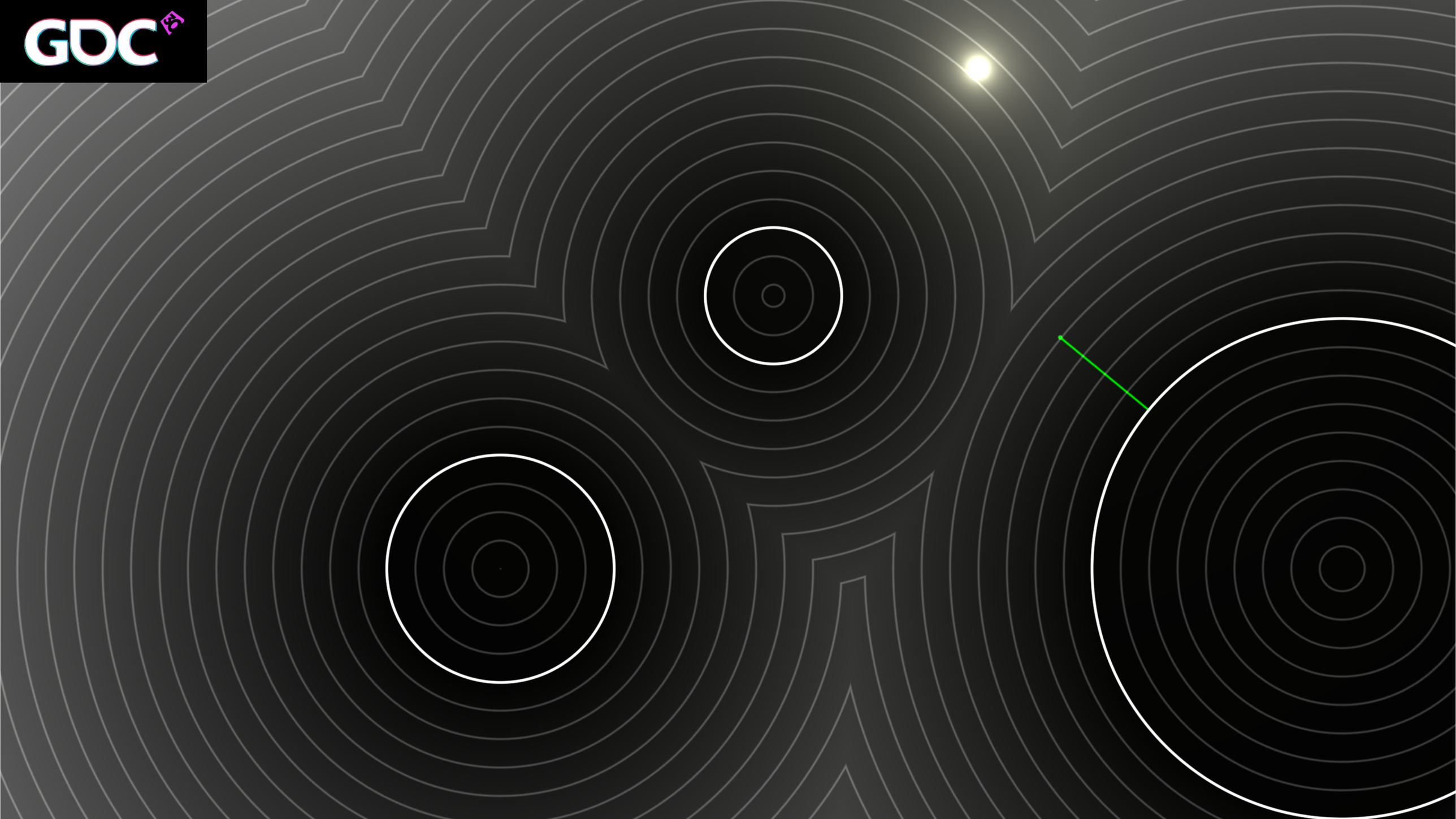

GDC

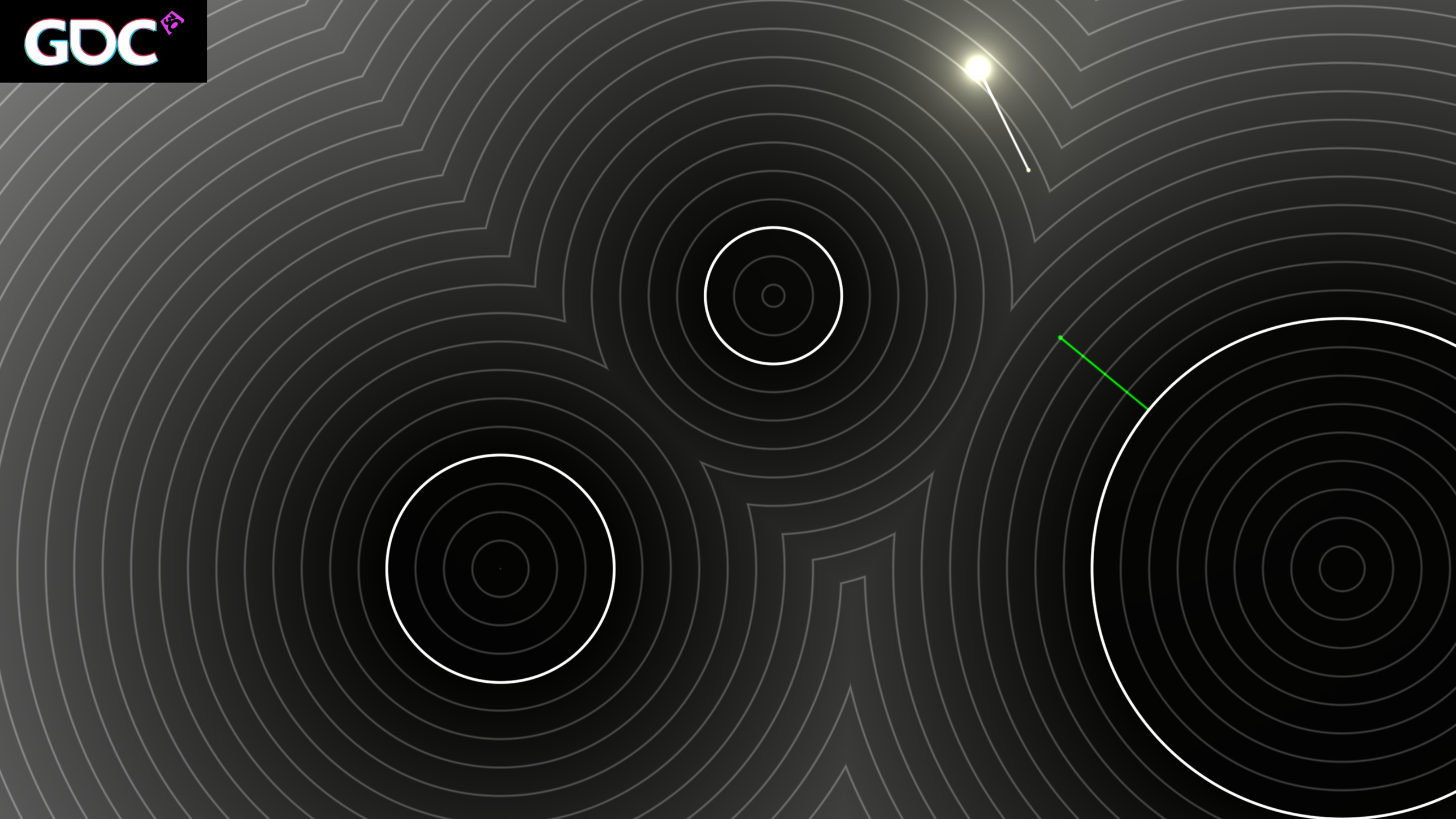


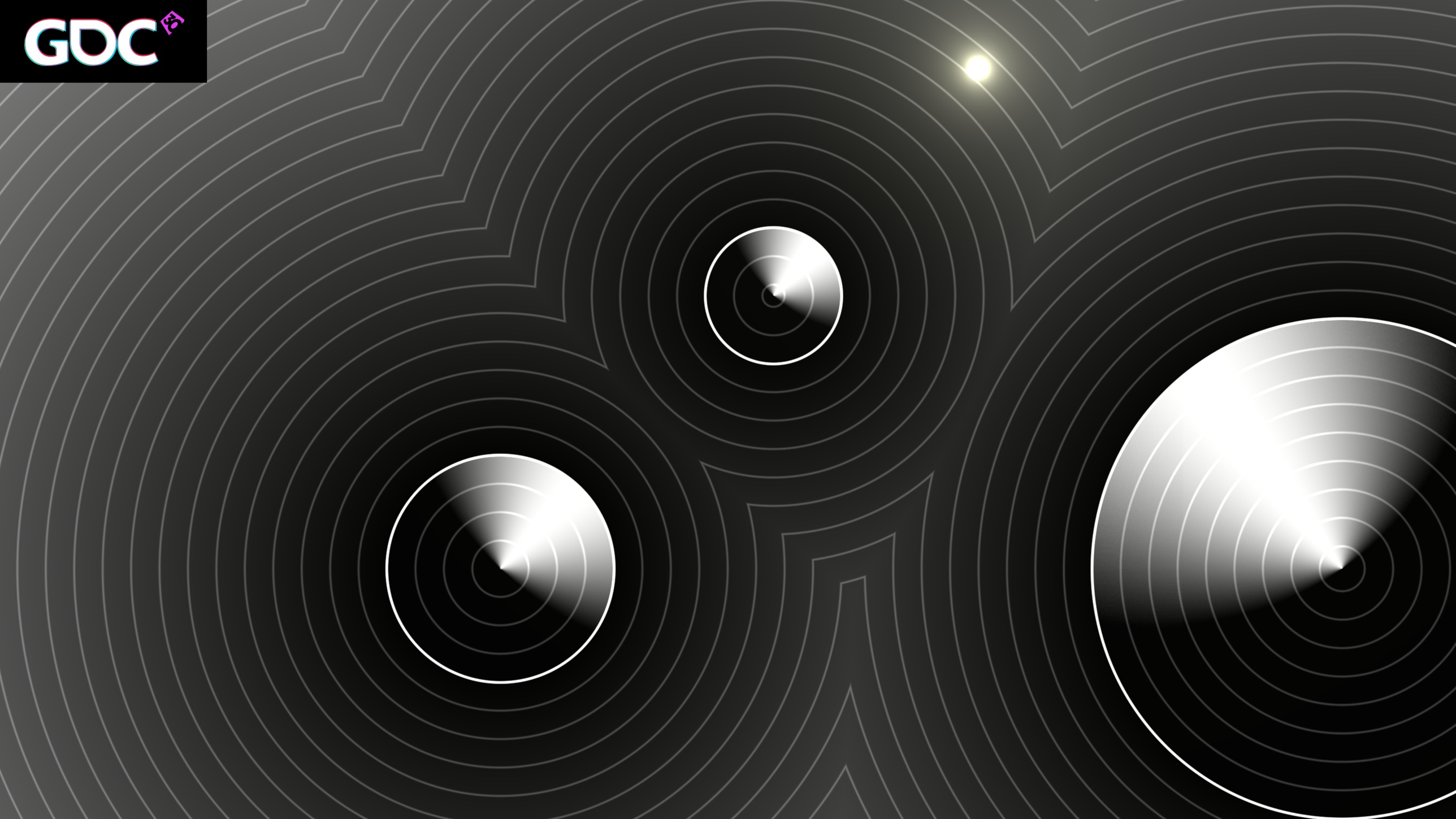


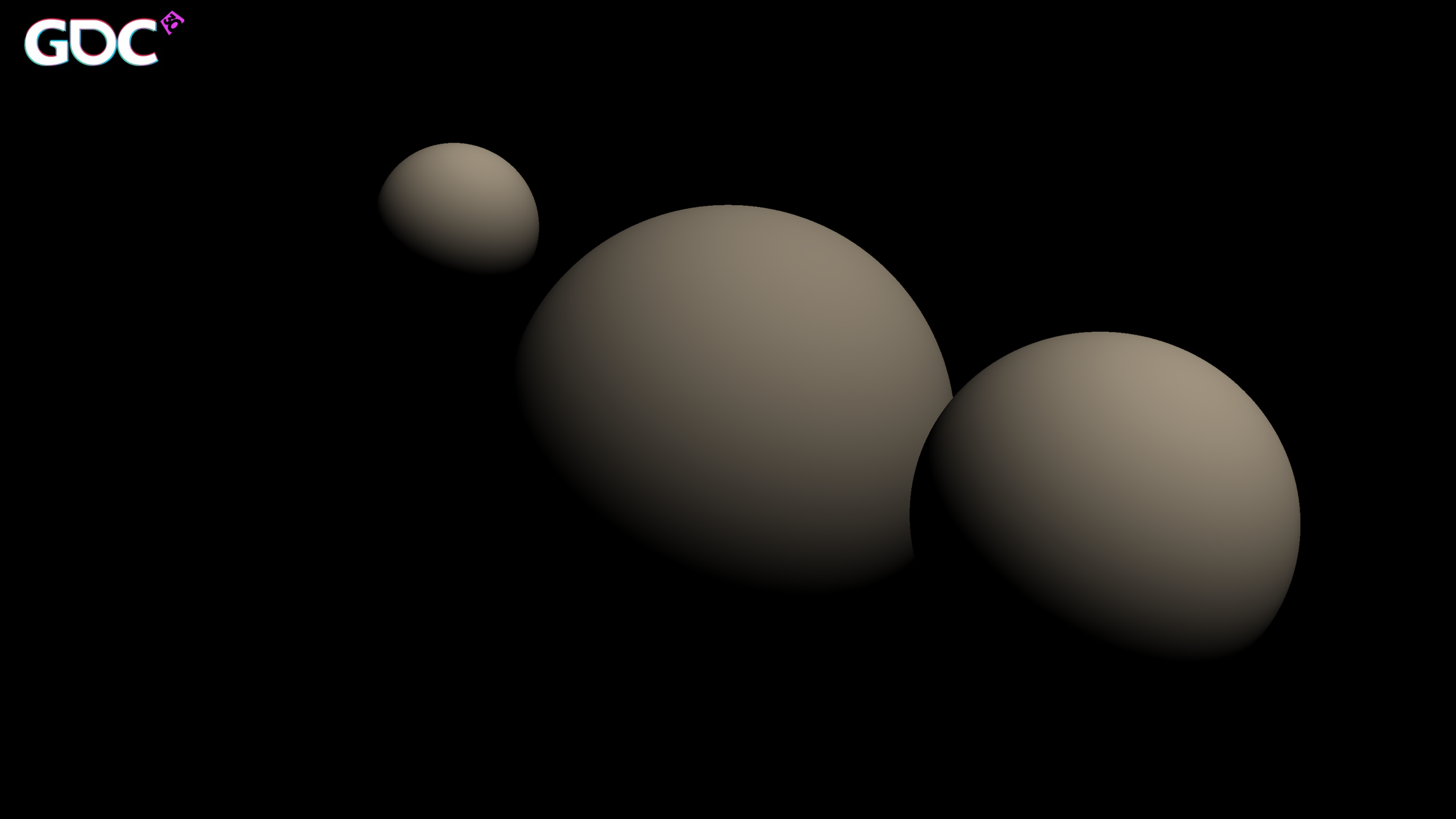
# Lights

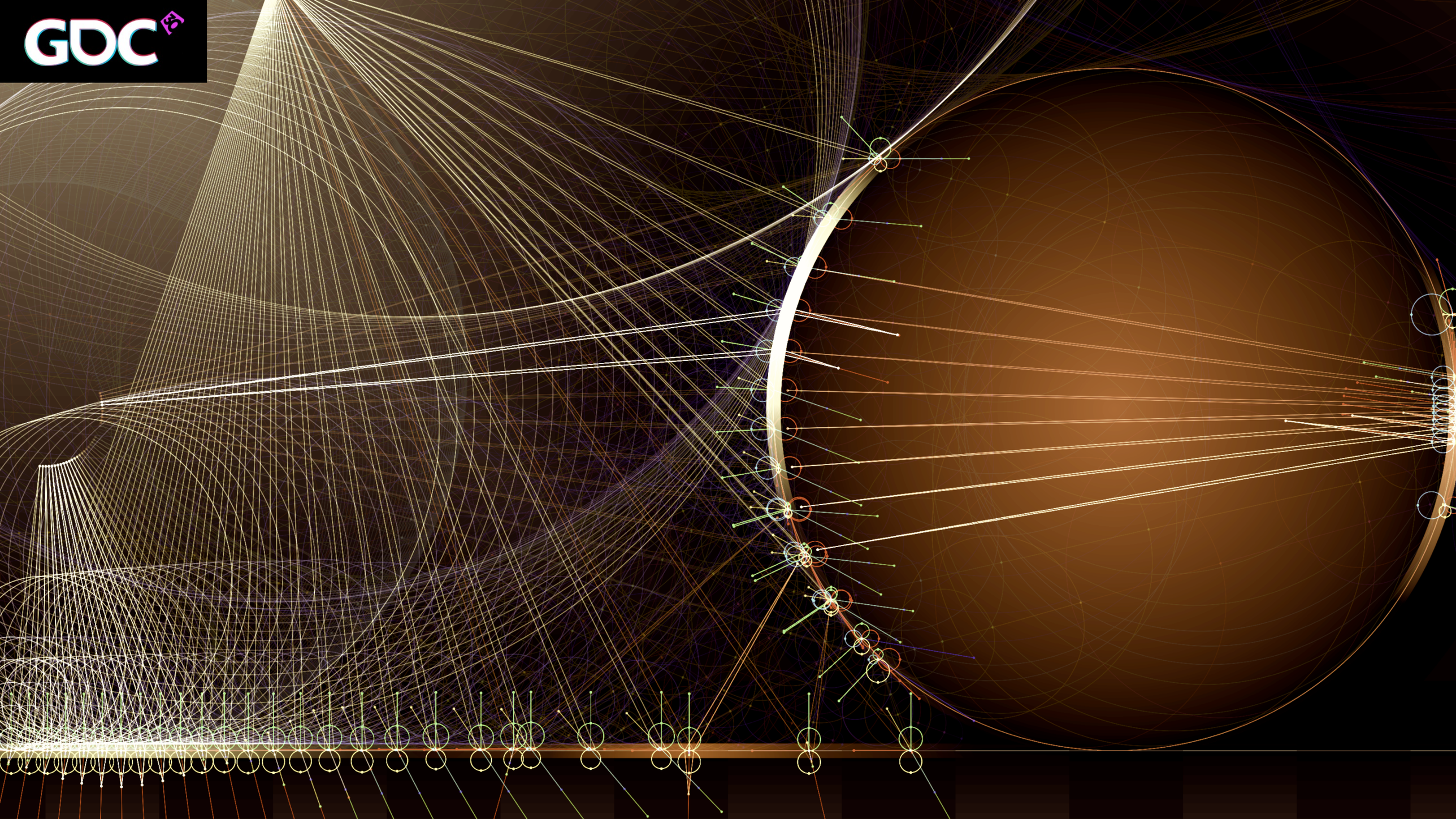


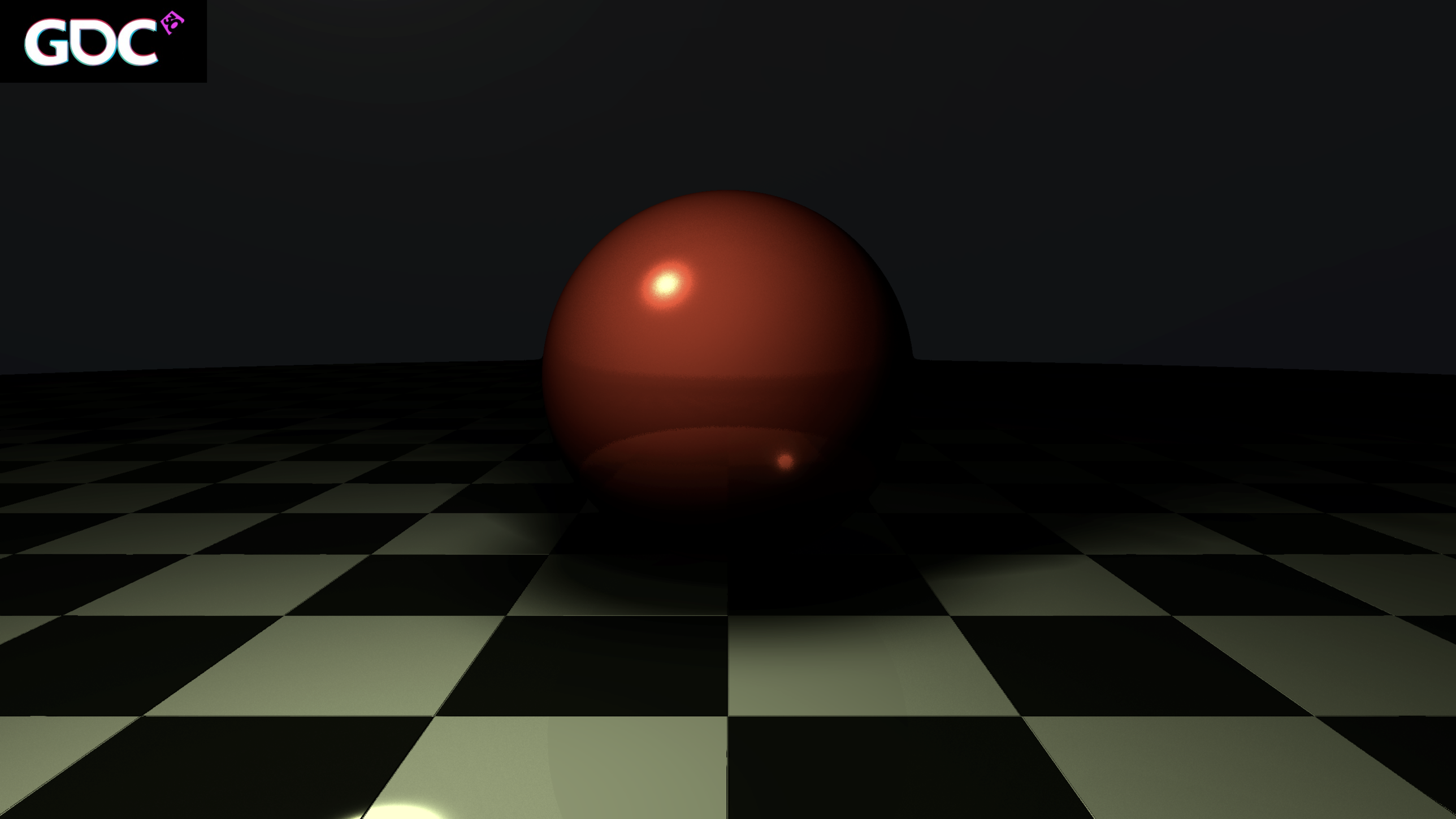


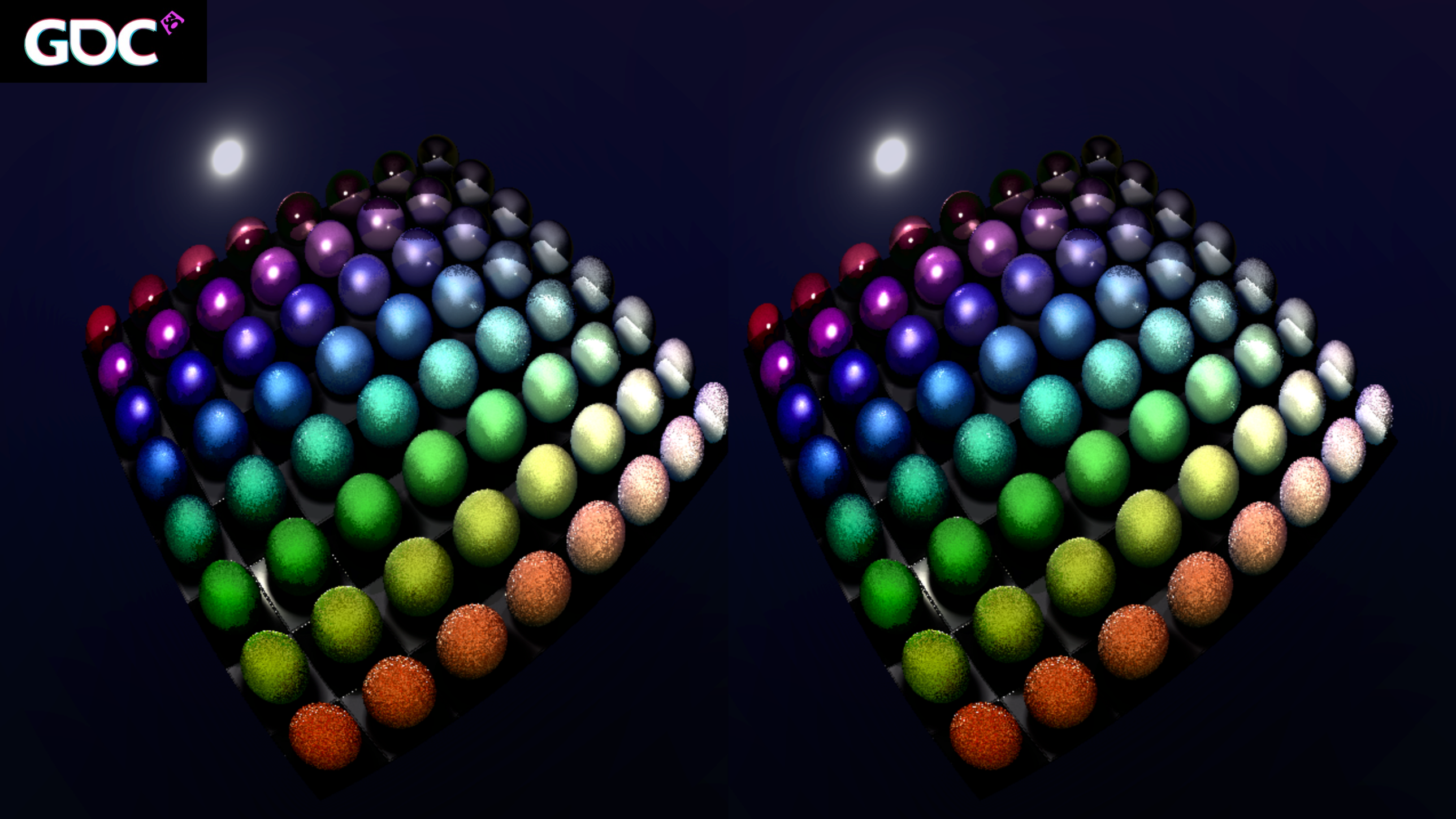

GDC


GDC




GDC




GDC