Digital konst

Av Peter Leinonen
http://leinonen.se

Demoscenen, WebGL och 3D-grafik med endast två trianglar

Vem är jag?

Namn: Peter Leinonen

Titel: Frontend- utvecklare på Evolve Technology

Just nu: React & Redux på Toyota Material Handling

Hobby: Javascript och shaders, samt elektronisk musik

E V
O L
V E

Digital konst?

Konst

“Uttrycket eller tillämpningen av mänsklig kreativ skicklighet och fantasi, vanligtvis i en visuell form som målning eller skulptur, som producerar verk som uppskattas främst för sin skönhet eller emotionella kraft.” - Wikipedia

Digital konst (Demos)

“Skapandet av dataprogram som genererar audiovisuella presentationer med hjälp av algoritmer och matematiska formler, som uppskattas främst för sin skönhet eller emotionella kraft.” - Peter

Demos (Intros)

- Självständiga, ofta extremt små, datorprogram som producerar audiovisuella presentationer.
- Syftet med ett demo är att visa skicklighet i programmering, bildkonst och musikaliska färdigheter.

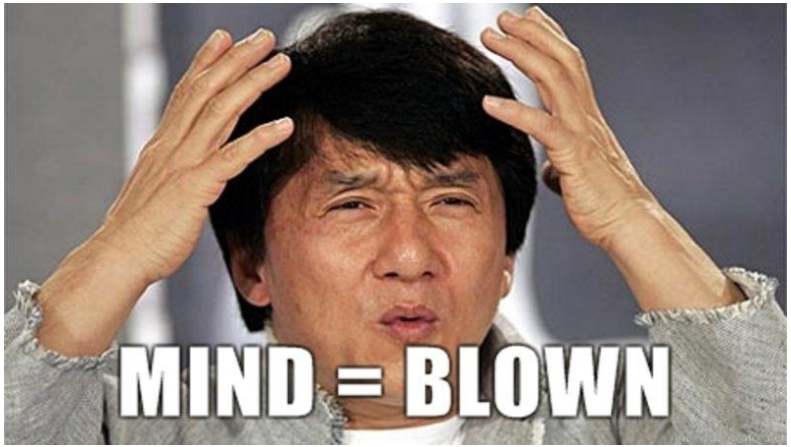
Fermi Paradox by Mercury (64K demo), 2016
https://www.youtube.com/watch?v=JZ6ZzJeWgpY

Demo - Dess beståndsdelar

- En arsenal med grafiska effekter
- Musikspelare / mjukvarusynthesizer
- Musikdata

Hur får detta plats i 64KB ?

Hur får detta plats i 4KB ?



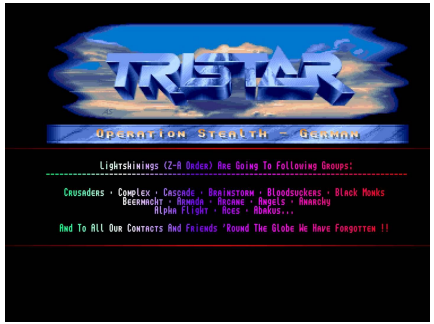
Historia

Sent 70-tal, hemdator (Atari, Amiga, C64, PC)

Crackade spel, ascii/ansi grafik, introskärmar

Utvecklades till en egen gren: demoscenen

Fokus på att göra cool grafik och musik



Demopartyn

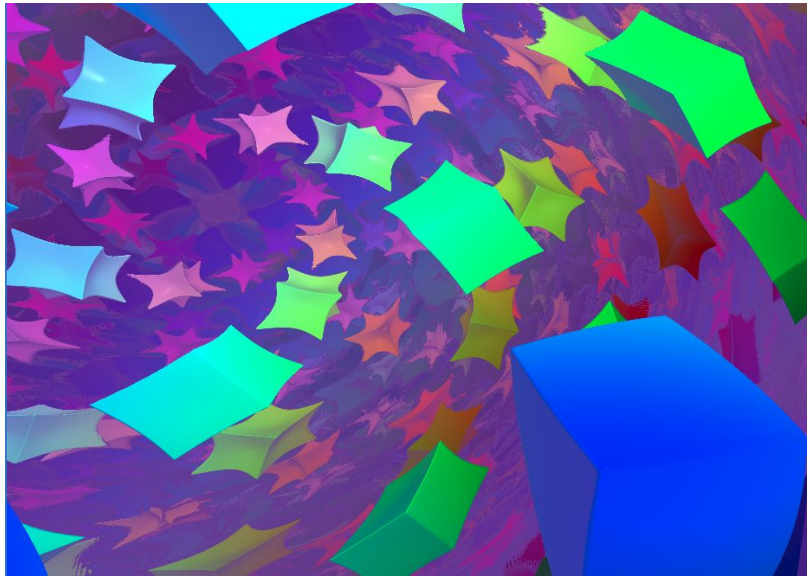
Grupperingar samlas och tävlar mot varandra. Vem kan göra det bästa demot?

Kategorier:

- “Friform” demos
- Storleksbegränsade demos
  - 64K intro (65536 bytes)
  - 4K intro (4096 bytes)

Assembly 2004 demo party, Finland.
Image from Wikipedia

Exempelprojekt

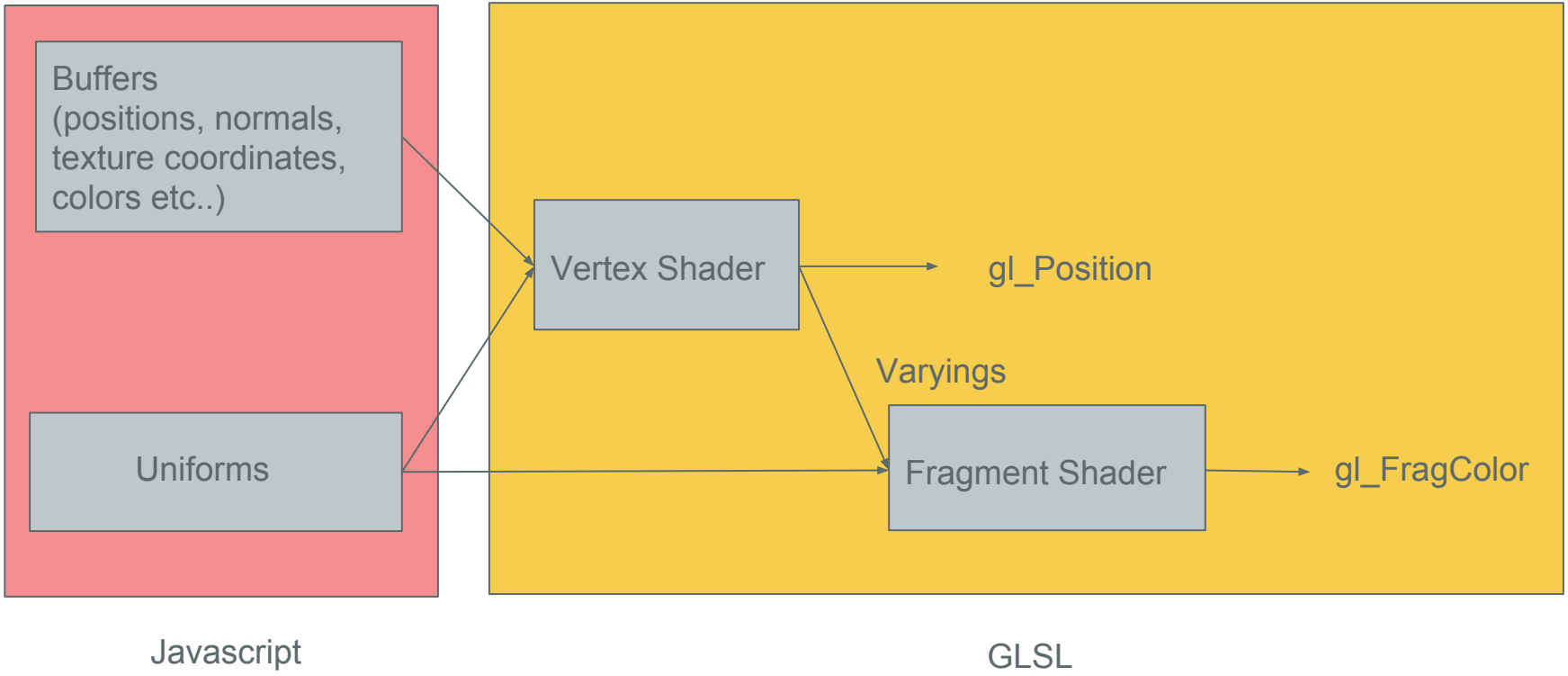


https://github.com/leinonen/dayoftheprogrammer

WebGL (Web Graphics Library)

- JavaScript API för 2D- och 3D-grafik i webbläsaren
- GPU-acceleration
- Kod för att styra shadern skrivs i Javascript
- Shaderkod skrivs i GLSL (GL Shading Language)

WebGL overview



GLSL Crash Course

Datatyper

`int, float,`
`vec2, vec3,`
`vec4, mat2,`
`mat3, mat4,`
`struct`

`uniform,`
`varying`

Inbyggda funktioner

`sin, cos, length,`
`normalize, dot,`
`cross, clamp,`
`texture2D,`
`texture3D`

Main metoden

Skriv ett värde till `gl_Position` eller `gl_FragColor`

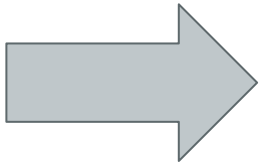
Två shaders blir ett program

Vertex Shader

```
attribute vec3 position;

void main() {
  gl_Position = vec4(position, 1);
}
```

Fragment Shader

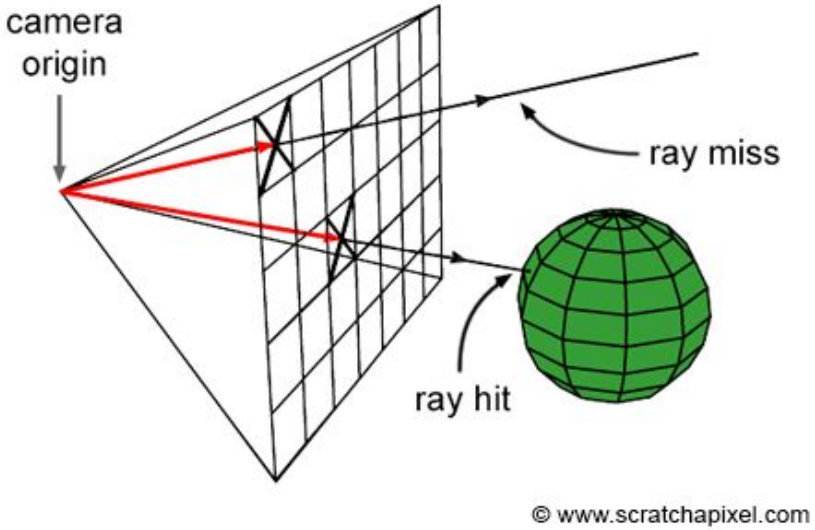
```
uniform vec2 resolution; // canvas size

void main() {
  vec2 p = gl_FragCoord.xy / resolution.xy;
  vec3 col = vec3(p.x, p.y, 0);
  gl_FragColor = vec4(col, 1);
}
```



Olika sätt att rendera

- Bygg all geometri mha polygoner
  - Kräver 3D-editor för att bygga komplexa objekt (spel)
  - Lätt att mappa texturer
  - Kräver många polygoner för komplexa objekt
  - WebGL är bra på att hantera stora mängder polygoner (hundratusentals)
- Bygg geometri procedurellt (också med polygoner)
  - Svårt att blenda olika objekt..
- Bygg geometri utan polygoner
  - Definera objekt matematiskt (raytracing)
  - Svårt att mappa texturer
    - Man kan generera texturer procedurellt också!
  - Behöver plotta resultatet pixel för pixel…. (fragment shader?)
- Går givetvis att mixa båda approacher…

Ray tracing



Vertex data - Två trianglar

```
let buffer = gl.createBuffer()
let vertices = new Float32Array([
  -1.0, -1.0,  1.0, -1.0, -1.0,  1.0,
  -1.0,  1.0,  1.0, -1.0,  1.0,  1.0
])

gl.bindBuffer(gl.ARRAY_BUFFER, buffer)

gl.bufferData(gl.ARRAY_BUFFER, vertices,
gl.STATIC_DRAW)

gl.drawArrays(gl.TRIANGLES, 0, 6)
```

Koordinatsystemet går från -1.0 till 1.0

3D-grafik, med två trianglar?

Definera objekt procedurellt istället för med polygoner

- Ray tracing
  - Traditionellt långsamt och prestandakrävande
  - GPU kan hjälpa oss boosta prestanda!
- Ray marching
  - Variant av ray tracing
- Fragment shadern gör alla beräkningar

Ray marched snail by Inigo Quilez
Ca 800 rader GLSL
https://www.shadertoy.com/view/ld3Gz2

Ray tracing vs Ray marching

Skillnaden ligger i hur objekt i scenen är definerade, vilket leder till olika metoder att hitta skärningspunkter

- Ray tracing
  - Analytisk (geometrisk) lösning av ekvationer för att hitta skärningspunkter
  - Svårt att beräkna för komplexa objekt
- Ray marching
  - Trial and error, evaluera SDF
  - “Are we there yet?”
  - Kvaliteten beror på antal iterationer (steg), prestanda vs kvalitet

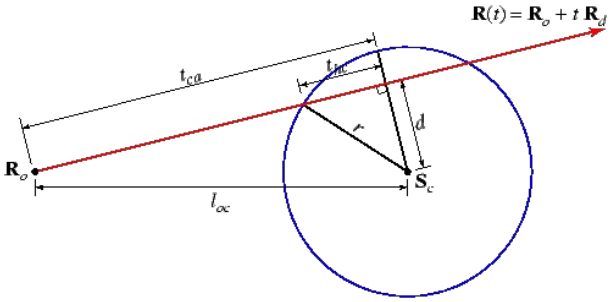


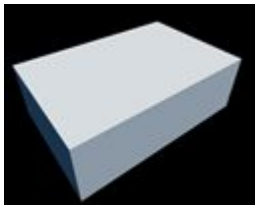
Image from Google..

Signed Distance Functions (SDF)

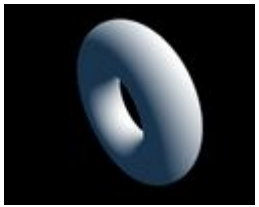
- Funktion som *beräknar kortaste avståndet* mellan en punkt i rummet och någon yta/volym som representeras av funktionen
- Signumet (positivt eller negativt) av resultatet indikerar om punkten är *innanför eller utanför*.

Sfär

Kub/box



Torus / munk



Plan

Images by Inigo Quilez

Ray marching

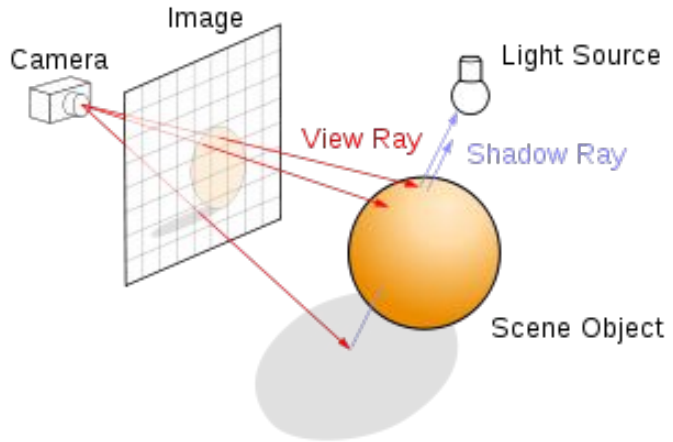
ro = ray origin, rd = ray direction, Δt = step size

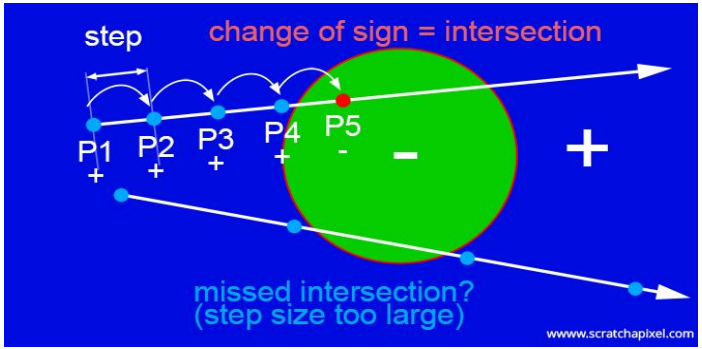
Algoritmen

$r(k) = ro + \Delta t\ k\ rd$

```
1: k = 0
2: d = 0
3: while k < kmax do
4:    d = f(r(k))
5:    if d ≤ 0 then return kΔt   // träff
6:    k = k + 1
7: return 0                      // ingen träff
```

kmax = antal steg / iterationer

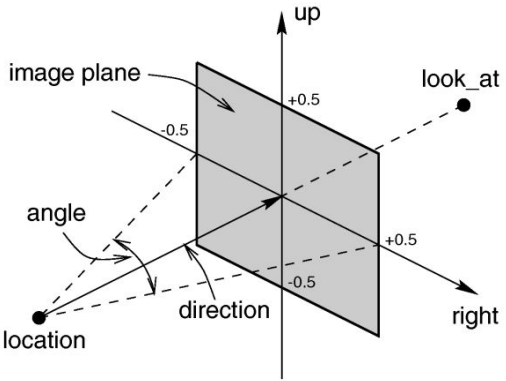


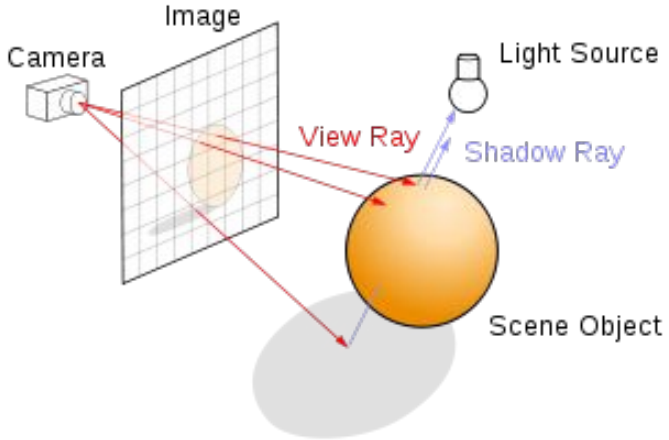


Images by Wikipedia and Scratchapixel

Ray direction

```
vec3 rayDirection(vec2 uv, vec3 camPos, vec3 lookAt, float fov) {
  vec3 forward = normalize(lookAt - camPos);
  vec3 right   = normalize(vec3(forward.z, 0., -forward.x ));
  vec3 up      = normalize(cross(forward, right));
  return normalize(forward + fov*uv.x*right + fov*uv.y*up);
}
```

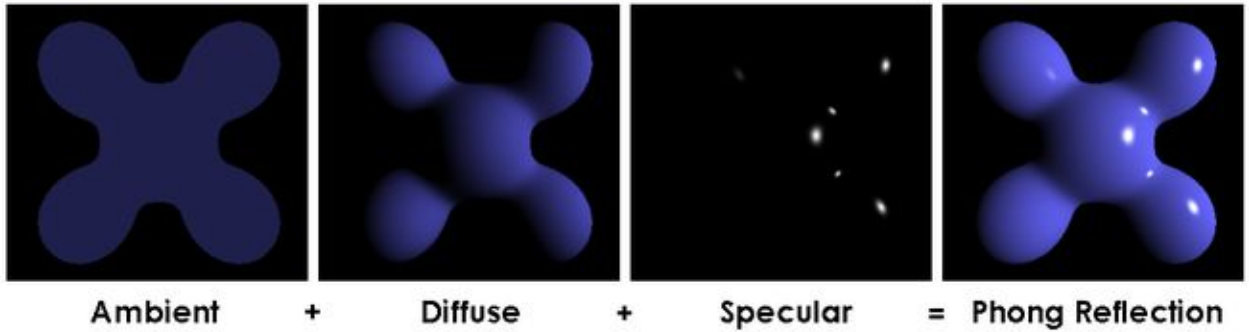




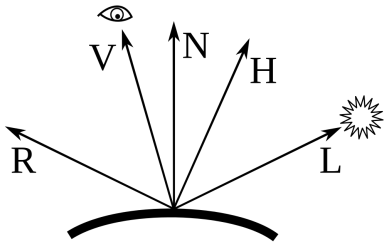
Ljus & Material - Phong Shading/Reflection

Ljusmodell utvecklad av *Bui Tuong Phong 1975*

- Ambient
- Diffuse
- Specular



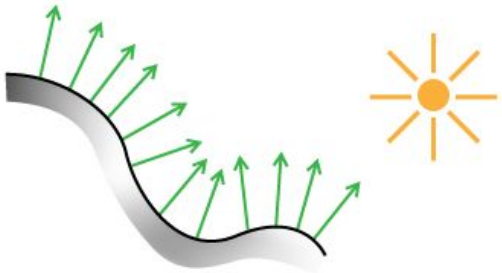
$$I_\mathrm{p} = k_\mathrm{a} i_\mathrm{a} + \sum_{m \in \text{ lights}} (k_\mathrm{d} (\hat{L}_m \cdot \hat{N}) i_{m,\mathrm{d}} + k_\mathrm{s} (\hat{R}_m \cdot \hat{V})^\alpha i_{m,\mathrm{s}}).$$



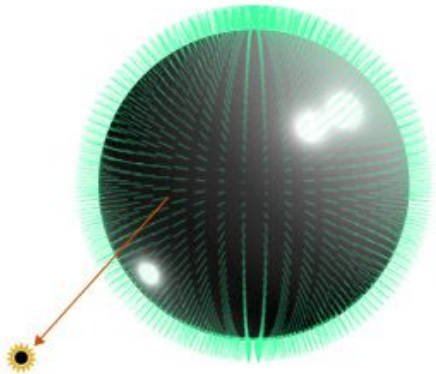
Hur beräknas normalen?

Man kan beräkna normalen genom att sampla punkter runt omkring nuvarande punkt. En slags förenklad derivata. ε är ett litet tal, *f(x,y,z)* är SDF-funktionen

$$\vec{n} = \begin{bmatrix} f(x+\varepsilon, y, z) - f(x-\varepsilon, y, z) \\ f(x, y+\varepsilon, z) - f(x, y-\varepsilon, z) \\ f(x, y, z+\varepsilon) - f(x, y, z-\varepsilon) \end{bmatrix}$$

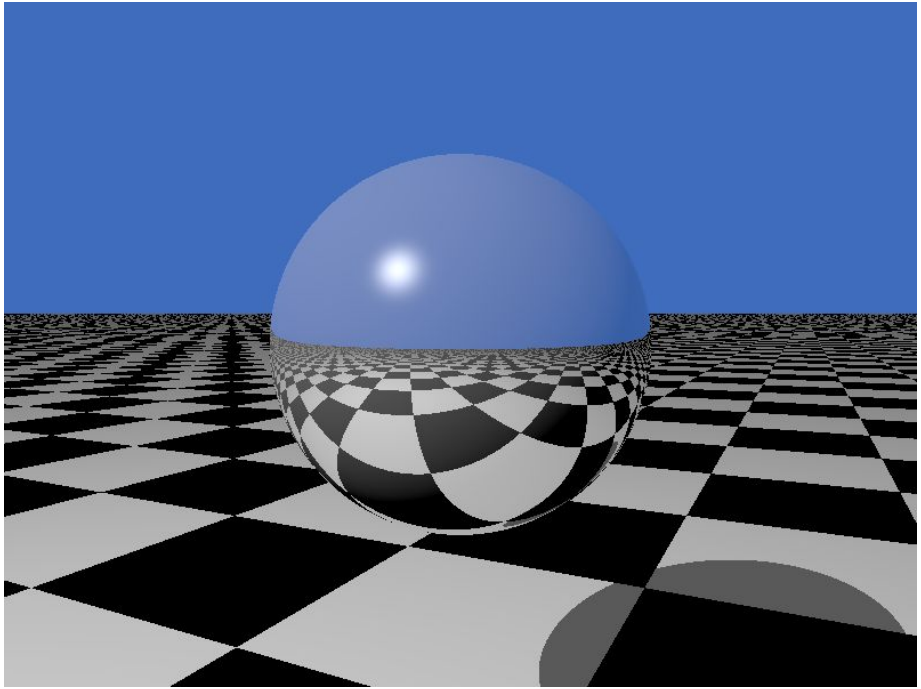


Images from Google



Reflektion

- Beräkna reflektionsvektor
  - vec3 rdRef = reflect(rd, normal);
- Raymarcha igen
- Beräkna färg och ljus
- Addera till föregående resultat



Translation, skalning och rotation

Translation / Positionering

```
vec3 offs = vec3(0, 6, 0);

float result = sdCube(p + offs);
```

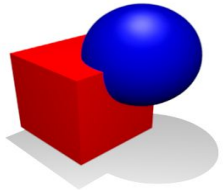
Skalning

```
float s = 6.0;

float result = sdCube(p / s) * s;
```

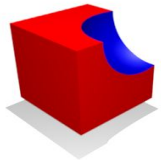
Rotation

```
mat2 rot2( float angle ) {
  float c = cos( angle );
  float s = sin( angle );
  return mat2( c, s,-s, c);
}

p.xy *= rot2(45. * PI);
p.xz *= rot2(45. * PI);

float result = sdCube(p);
```

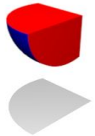
CSG - Constructive Solid Geometry



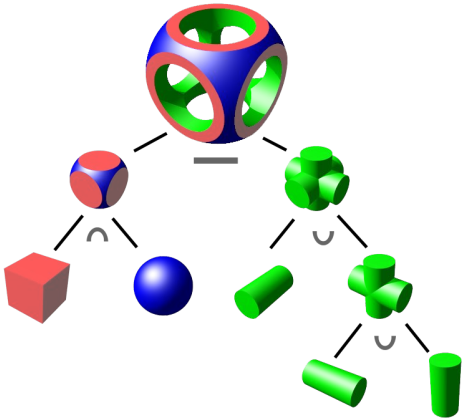
Union (∪)



Difference (−)



Intersection (∩)



Olika material

Packa resultatet av SDF funktionen som en vec2, där x komponenten är resultatet, och y komponenten är ett material-ID.

Ex:

```
vec3 result = vec2(sdCube(p, vec3(1.0)), BOX_ID);

result = sdDifference(result, vec2(sdSphere(p, 1.0), SPHERE_ID);
```

Domain repetition

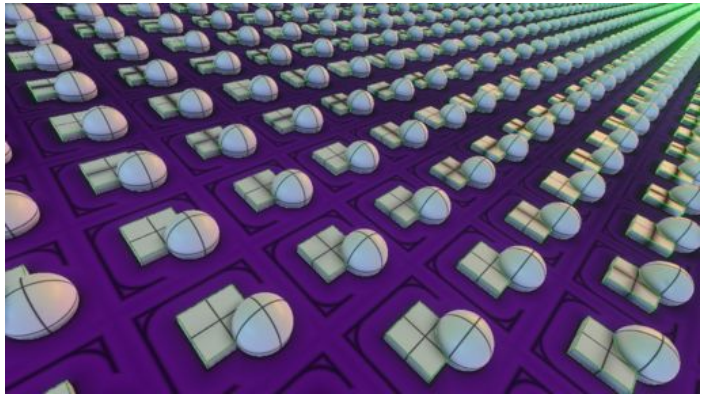
Modulo operator

Kan repetera i x-led, y-led, z-led, eller alla tre

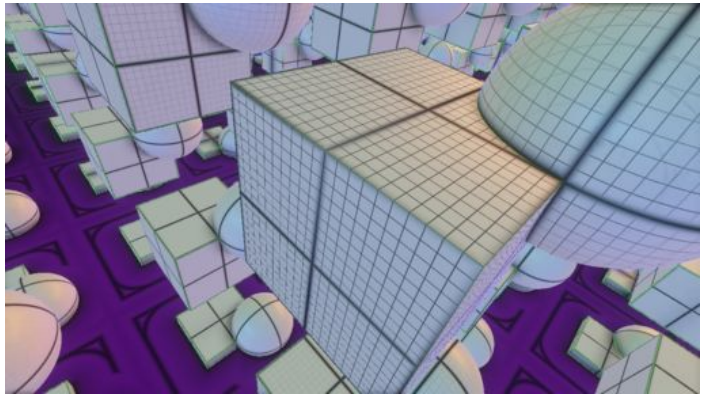
SDF-objektet måste få plats, annars uppstår distortion, och allt ser konstigt ut

```
p = mod(p, 2.0) - 1.0;
```

2.0 är storleken på “domänen”, och -1.0 centrerar koordinatsystemet i domänen

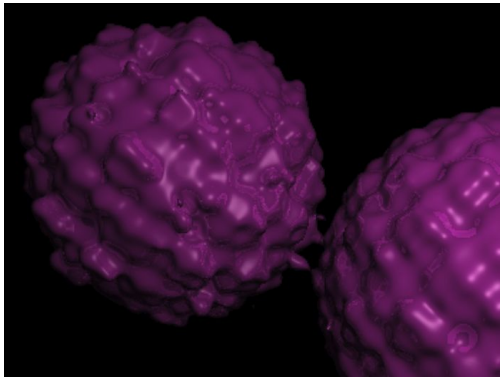


Bilder från Mercury’s SDF bibliotek

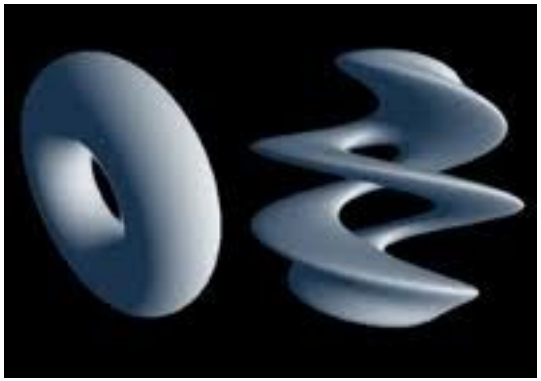


Deformering och distortion

```
float bump(vec3 p) {
  return .5 + .5 *
    sin(p.x*PI*2.) *
    cos(p.y*PI*2.) *
    sin(p.z*PI*2.);
}
…

float b = bump(p) * .03;

float result = sdSphere(p + b);
```



“2dmix” by Mikael Stalvik



Twisting using the y-position as an angle

```
p.xz *= rot2(p.y * PI * .3);
```

Prestanda / optimering

- Använd FPS Meter i Chrome (developer tools), för att se framerate
  - Även användbart för “debugging” av slö kod
- Undvik for-loopar
- Undvik onödiga if-satser (branching kan ge sämre prestanda)
- Skala canvas (men stretcha ut den)
  - Sämre kvalitet men bättre prestanda.
  - 0.5x, 0.25x

Att bygga ett demo

- Tar tid, massor med tid!
- Effekter kan kombineras…
  - Render to texture
- Behöver musik (WebAudio)
  - Synkronisera effekter till musiken
  - Kan få FFT data från WebAudio
- **Det är roligt och lärorikt!**
- **Det är mycket beroendeframkallande!**

Inspiration

Elevated by RGBA & TBC (Breakpoint 2009) 3 min 35 sek

- Demo som använder ray marching (dock ej WebGL, men tekniken är densamma)
- Endast 4kb ! (inklusive musik)

- https://www.youtube.com/watch?v=jB0vBmiTr6o

Shaderlab (Self promotion)

http://shaderlab.se

Shadertoy

http://shadertoy.com

Tack för att ni lyssnade!