Компьютерная графика
WebGL

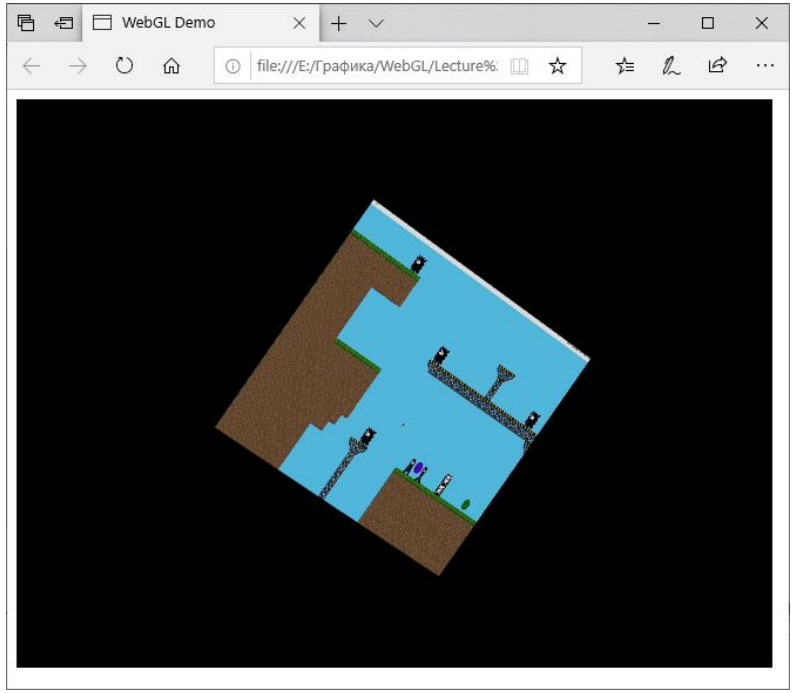
Лекция 10

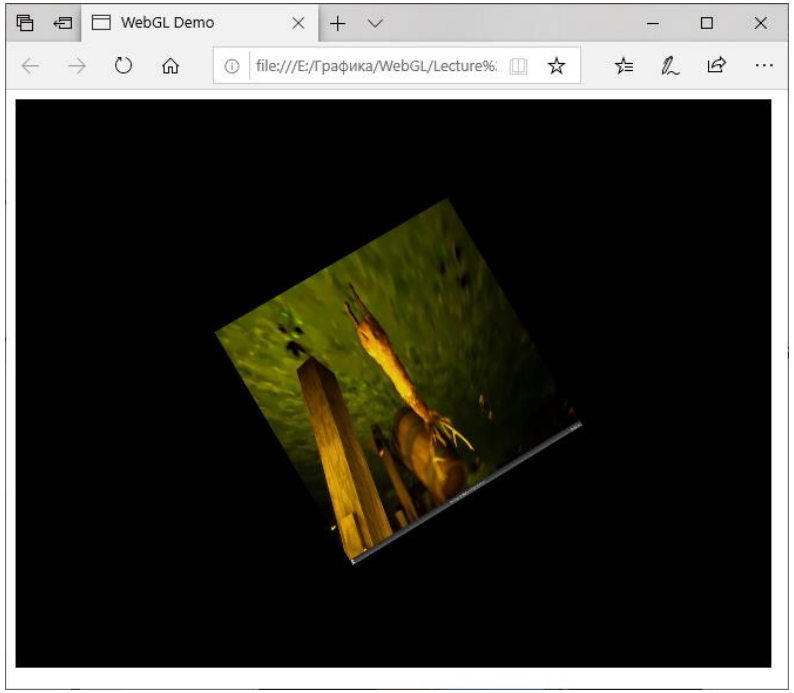
Демяненко Я.М. ЮФУ 2025

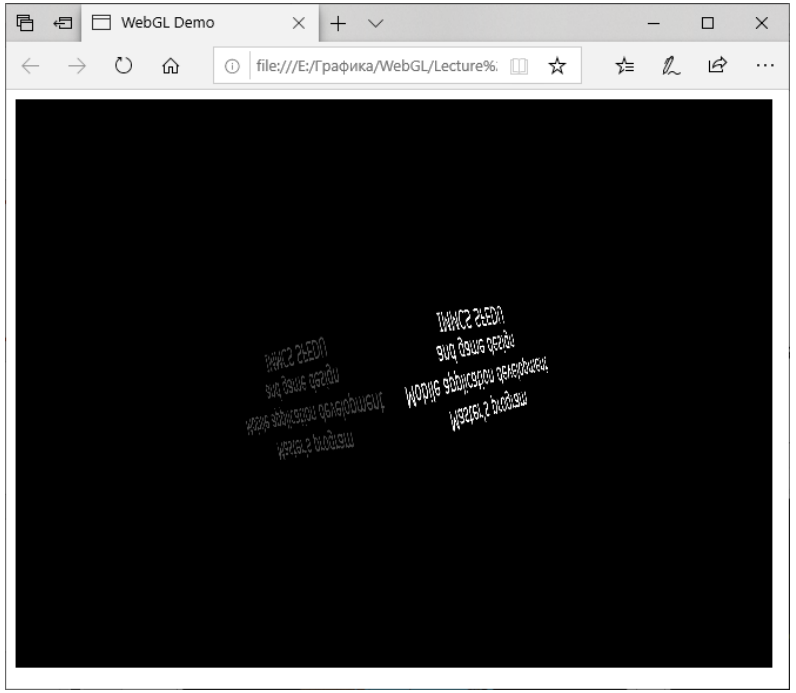
Анимация текстур в WebGL

Получение доступа к видео

Использование фреймов (видеокадров) в качестве текстуры







Получение доступа к видео

```
var copyVideo = false;

function setupVideo(url) {
  const video = document.createElement('video');

  var playing = false;
  var timeupdate = false;

  video.autoplay = true;
  video.muted = true;
  video.loop = true;

 video.addEventListener('playing', function() {
     playing = true;
     checkReady();
  }, true);

  video.addEventListener('timeupdate', function() {
     timeupdate = true;
     checkReady();
  }, true);

  video.src = url;
  video.play();

  function checkReady() {
    if (playing && timeupdate) {
      copyVideo = true;
    }
  }

  return video;
}
```

Использование видеокадров в качестве текстуры

```
function initTexture(gl) {
  const texture = gl.createTexture();
  gl.bindTexture(gl.TEXTURE_2D, texture);

  const level = 0;
  const internalFormat = gl.RGBA;
  const width = 1;
  const height = 1;
  const border = 0;
  const srcFormat = gl.RGBA;
  const srcType = gl.UNSIGNED_BYTE;
  const pixel = new Uint8Array([0, 0, 255, 255]);
  gl.texImage2D(gl.TEXTURE_2D, level, internalFormat, width, height, border, srcFormat, srcType, pixel);

  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);

  return texture;
}
```

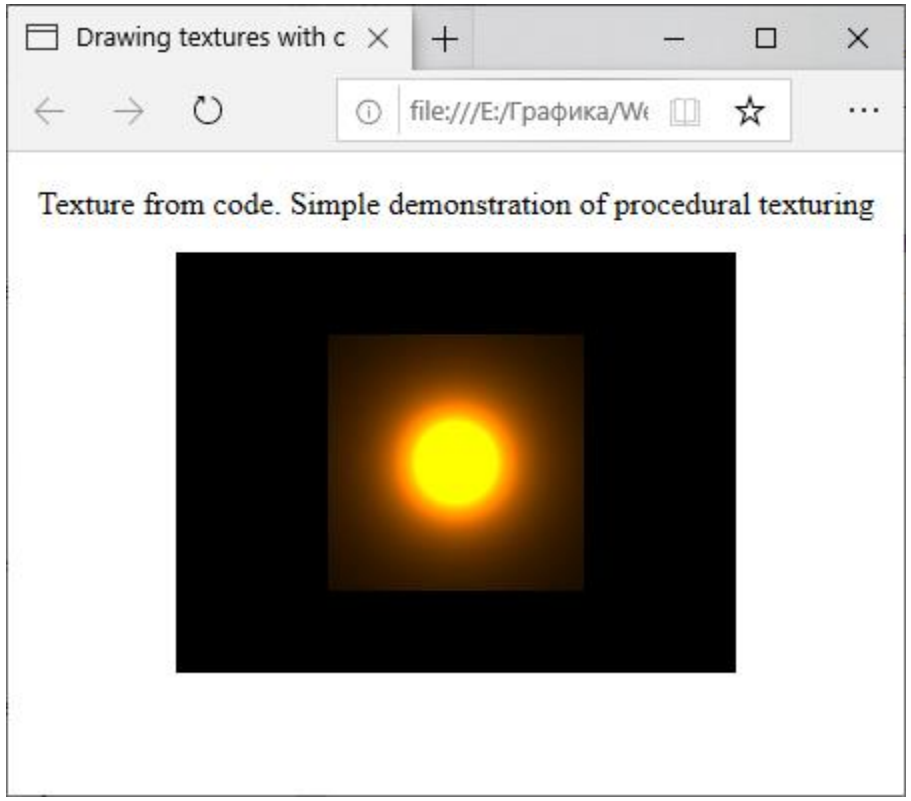
Обновление текстуры

```
function updateTexture(gl, texture, video) {
  const level = 0;
  const internalFormat = gl.RGBA;
  const srcFormat = gl.RGBA;
  const srcType = gl.UNSIGNED_BYTE;
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texImage2D(gl.TEXTURE_2D, level, internalFormat, srcFormat, srcType, video);
}
```

Часть функции main()

```
const texture = initTexture(gl);
const video = setupVideo('Firefox.mp4');
var then = 0;

function render(now) {
   now *= 0.001;  // в секунды
   const deltaTime = now - then;
   then = now;

   if (copyVideo) {
     updateTexture(gl, texture, video);
   }

   drawScene(…);
   requestAnimationFrame(render);
}
requestAnimationFrame(render);
```

Процедурные текстуры. Что хотим получить?

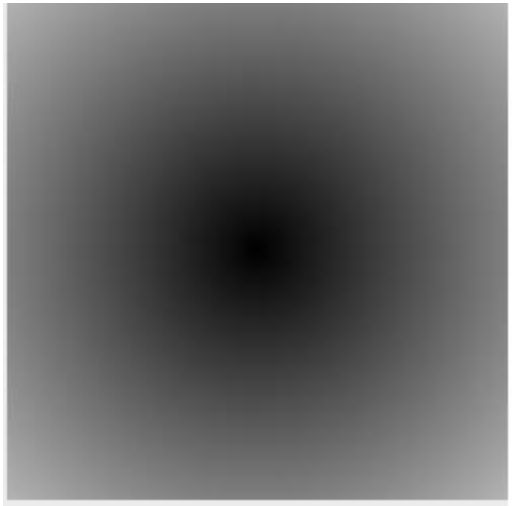


Текстурирование точечного спрайта с попиксельными вычислениями во фрагментном шейдере.

Фрагментный шейдер, описывающий круг

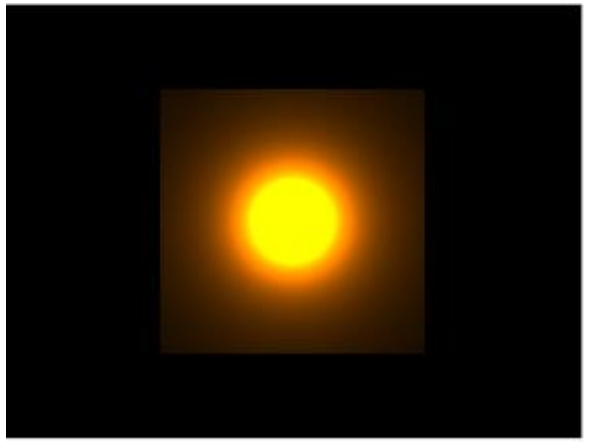
```
float radius = 20.;
vec2 center = vec2(100., 100.);

void main() {
  float distanceToCenter = distance(gl_FragCoord.xy, center);
  float inCircle = float(distanceToCenter < radius);
  gl_FragColor = vec4(inCircle);
}
```

```
uniform vec2 u_resolution;

void main(){
    vec2 st = gl_FragCoord.xy/u_resolution;
    float pct = 0.0;

    // a. The DISTANCE from the pixel to the center
    pct = distance(st,vec2(0.5));

    // b. The LENGTH of the vector  from the pixel to the center
    // vec2 toCenter = vec2(0.5)-st;
    // pct = length(toCenter);

    // c. The SQUARE ROOT of the vector  from the pixel to the center
    // vec2 tC = vec2(0.5)-st;
    // pct = sqrt(tC.x*tC.x+tC.y*tC.y);

    vec3 color = vec3(pct);

   gl_FragColor = vec4( color, 1.0 );
}
```



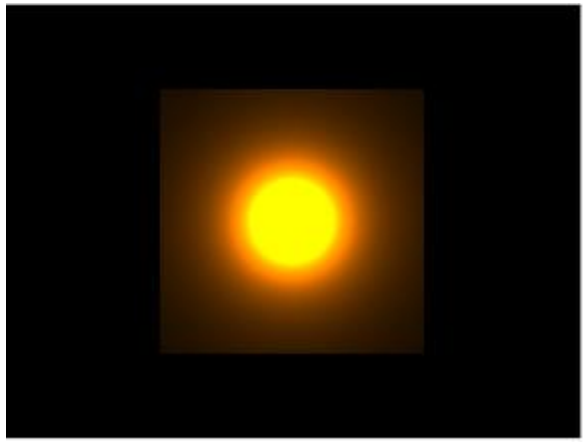
Вершинный шейдер

```
precision highp float;

attribute vec2 position;

void main() {
  gl_Position = vec4(position, 0.0, 1.0);
  gl_PointSize = 128.0;
}
```

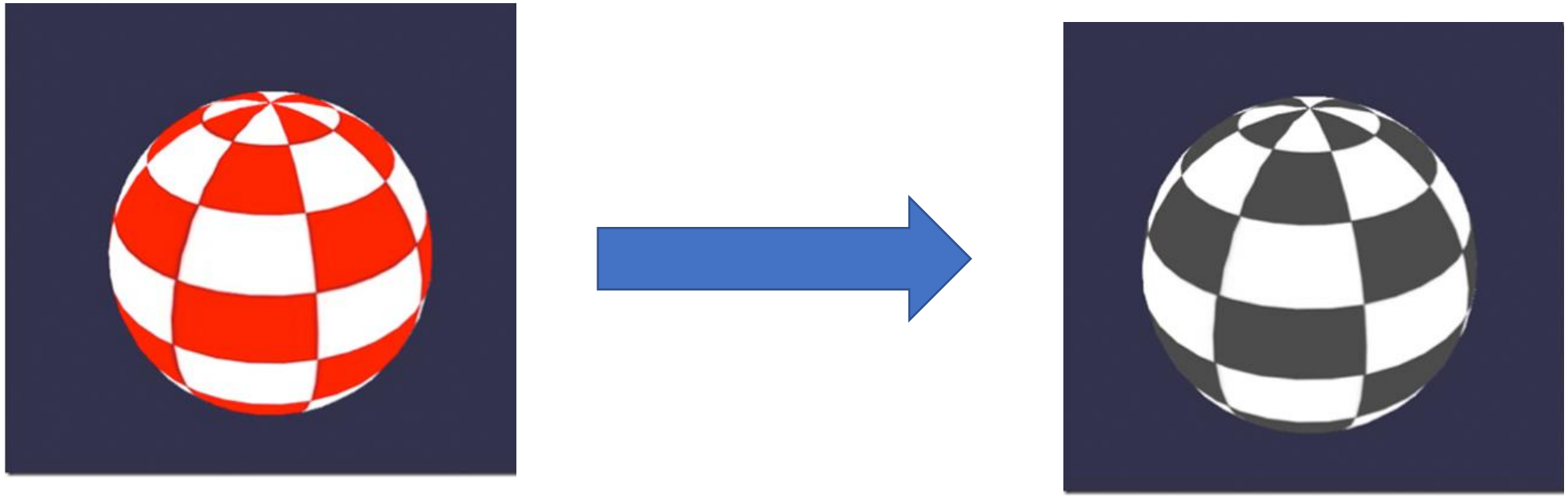


Фрагментный шейдер

```
precision mediump float;
void main() {
  vec2 fragmentPosition = 2.0*gl_PointCoord - 1.0;
  float distance = length(fragmentPosition);
  float distanceSqrd = distance * distance;
  gl_FragColor = vec4(
    0.2/distanceSqrd,
    0.1/distanceSqrd,
    0.0, 1.0 );
}
```



Что хотим получить?



Вершинный шейдер

```
precision mediump float;

attribute vec3 position;
attribute vec2 uv;

uniform mat4 worldViewProjection;

out vec2 vUV;

void main(void) {
   gl_Position = worldViewProjection * vec4(position, 1.0);

   vUV = uv;
}
```

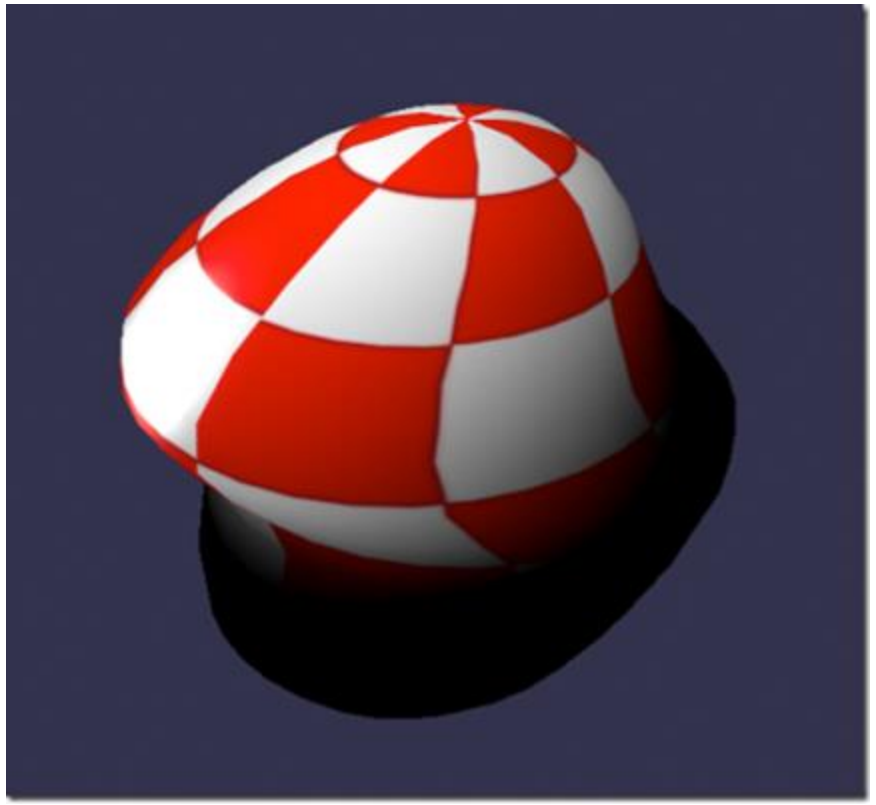
Фрагментный шейдер — версия 1

```
precision mediump float;

varying vec2 vUV;

uniform sampler2D textureSampler;

void main(void) {
   gl_FragColor = vec4(texture2D(textureSampler, vUV).ggg, 1.0);
}
```

Фрагментный шейдер — версия 2

```
precision mediump float;

in vec2 vUV;

uniform sampler2D textureSampler;

void main(void) {
   float luminance = dot(texture2D(textureSampler, vUV).rgb, vec3(0.3, 0.59, 0.11));
   gl_FragColor = vec4(luminance, luminance, luminance, 1.0);
}
```

Что хотим получить?

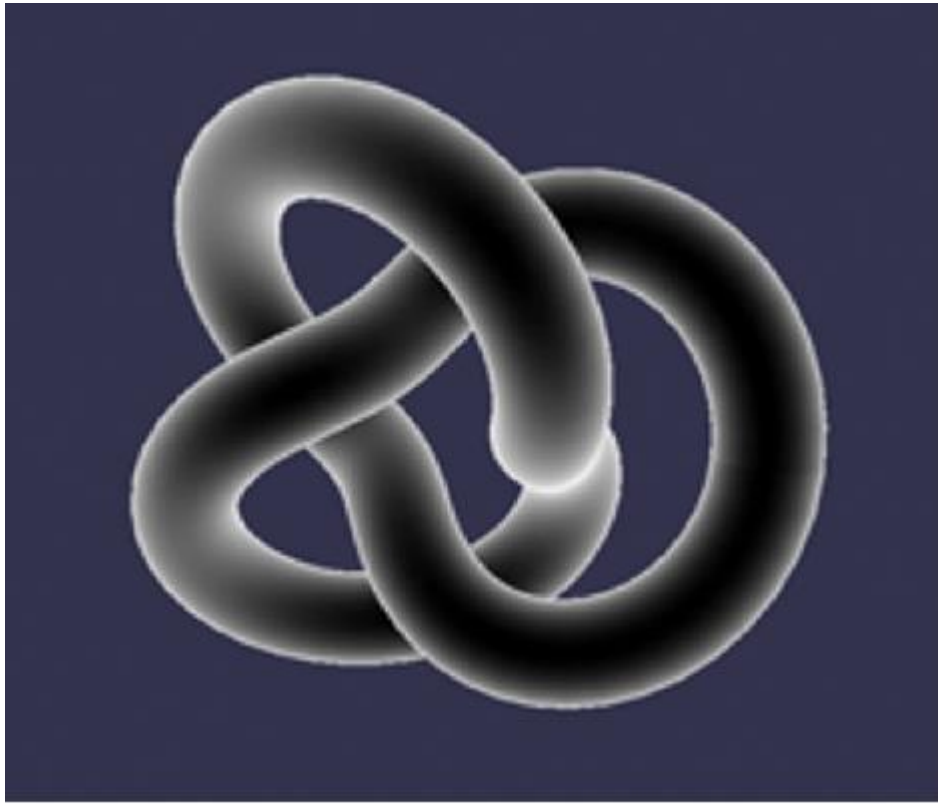


Вершинный шейдер

```
attribute vec3 position;
attribute vec3 normal;
attribute vec2 uv;

uniform mat4 worldViewProjection;
uniform float time;

out vec3 vPosition;
out vec3 vNormal;
out vec2 vUV;

void main(void) {
   vec3 v = position;
   v.x += sin(2.0 * position.y + (time)) * 0.5;

   gl_Position = worldViewProjection * vec4(v, 1.0);

   vPosition = position;
   vNormal = normal;
   vUV = uv;
}
```

Используем uniform-переменную time, чтобы получить динамические значения для генерирования волны

Что хотим получить?



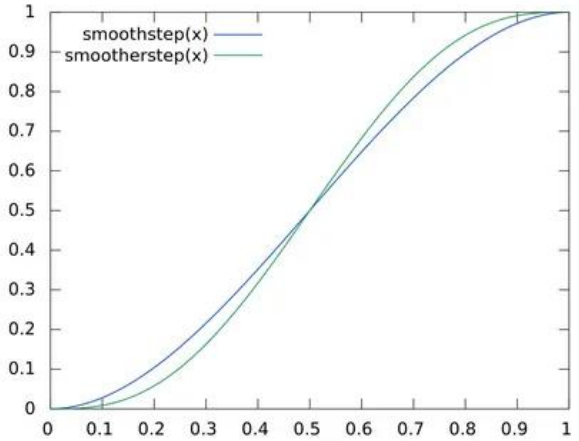
Формулы Френеля связывают амплитуды преломлённой и отражённой электромагнитных волн с амплитудой волны, падающей на плоскую границу раздела двух сред с разными показателями преломления.

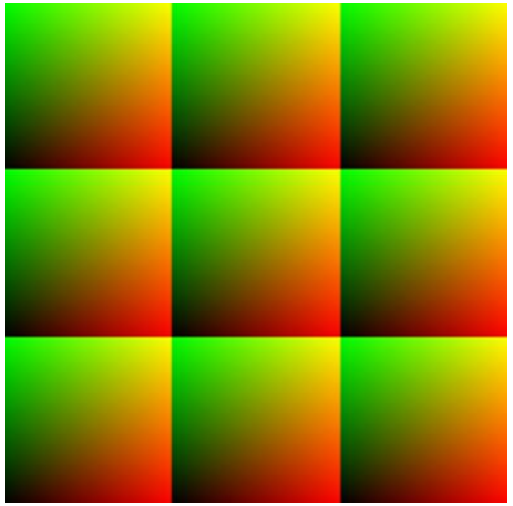
Для аппроксимации вклада фактора Френеля в зеркальное отражение используется аппроксимация Шлика.

Фрагментный шейдер

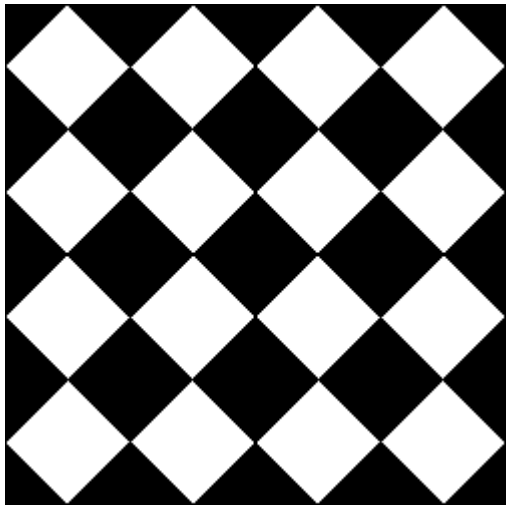
```
in vec3 vPositionW;
in vec3 vNormalW;

uniform vec3 cameraPosition;
uniform sampler2D textureSampler;

void main(void) {
    vec3 color = vec3(1., 1., 1.);
    vec3 viewDirectionW = normalize(cameraPosition - vPositionW);

    // Fresnel
    float fresnelTerm = dot(viewDirectionW, vNormalW);
    fresnelTerm = clamp(1.0 - fresnelTerm, 0., 1.);

    gl_FragColor = vec4(color * fresnelTerm, 1.);
}
```

```
uniform vec2 u_resolution;
uniform float u_time;

float circle(in vec2 _st, in float _radius){
    vec2 l = _st-vec2(0.5);
    return 1.-smoothstep(_radius-(_radius*0.01),
                         _radius+(_radius*0.01),
                         dot(l,l)*4.0);
}

void main() {
    vec2 st = gl_FragCoord.xy/u_resolution;
    vec3 color = vec3(0.0);

    st *= 3.0;      // Scale up the space by 3
    st = fract(st); // Wrap around 1.0

    // Now we have 9 spaces that go from 0-1

   color = vec3(circle(st,0.5));

   gl_FragColor = vec4(color,1.0);
}
```



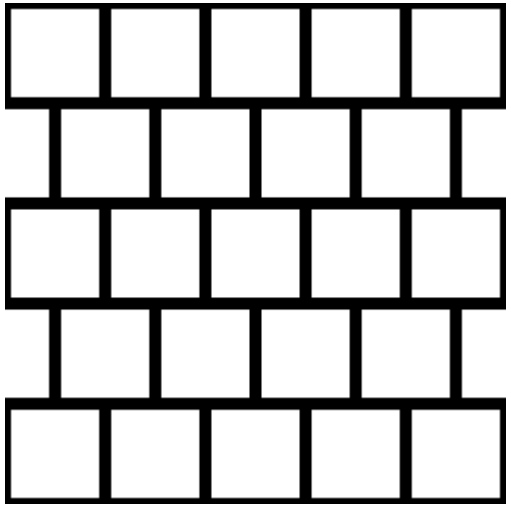


Что хотим получить?



```
uniform vec2 u_resolution;
uniform float u_time;

#define PI 3.14159265358979323846

vec2 rotate2D(vec2 _st, float _angle){
    _st -= 0.5;
    _st =  mat2(cos(_angle),-sin(_angle), sin(_angle),cos(_angle)) * _st;
    _st += 0.5;
    return _st;
}

vec2 tile(vec2 _st, float _zoom){
    _st *= _zoom;
    return fract(_st);
}

float box(vec2 _st, vec2 _size, float _smoothEdges){
    _size = vec2(0.5)-_size*0.5;
    vec2 aa = vec2(_smoothEdges*0.5);
    vec2 uv = smoothstep(_size,_size+aa,_st);
    uv *= smoothstep(_size,_size+aa,vec2(1.0)-_st);
    return uv.x*uv.y;
}

void main(void){
    vec2 st = gl_FragCoord.xy/u_resolution.xy;
    vec3 color = vec3(0.0);

    // Divide the space in 4
    st = tile(st,4.);

    // Use a matrix to rotate the space 45 degrees
    st = rotate2D(st,PI*0.25);

    // Draw a square
    color = vec3(box(st,vec2(0.7),0.01));

    gl_FragColor = vec4(color,1.0);
}
```

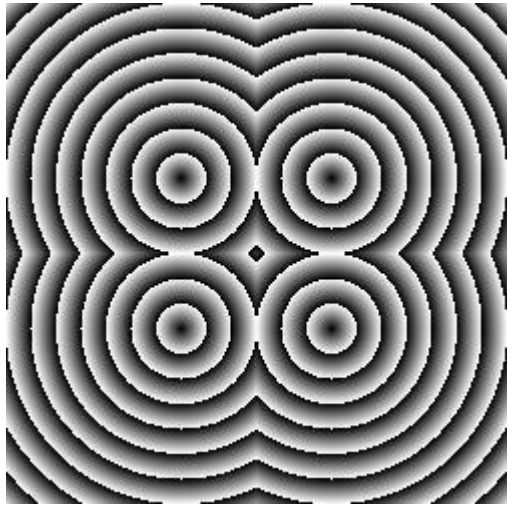
Что хотим получить?



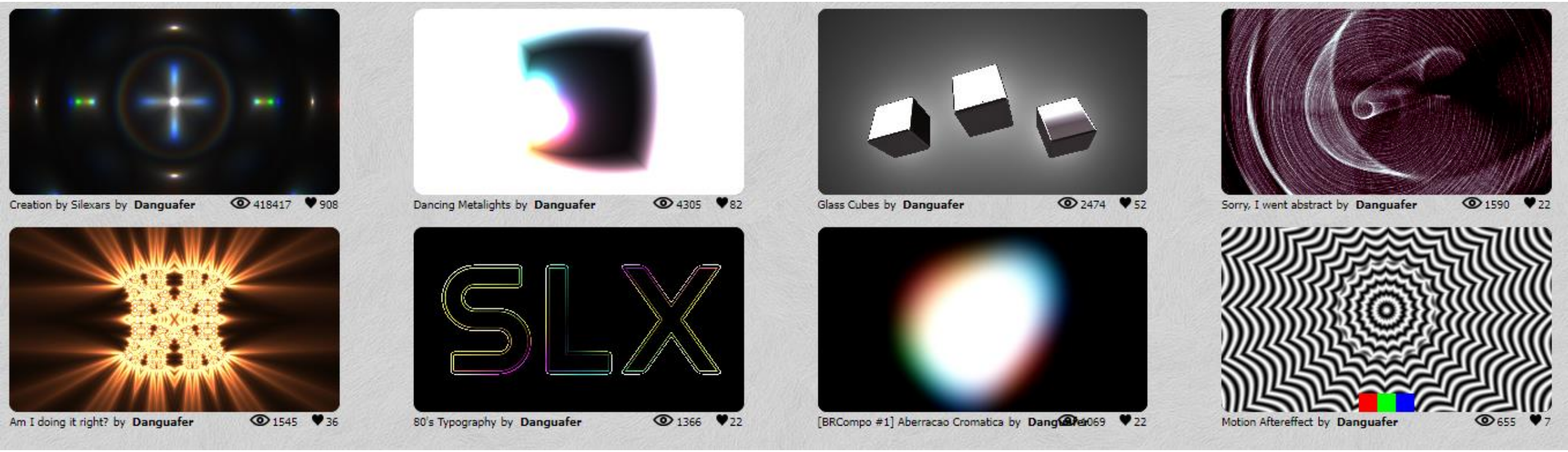
```
uniform vec2 u_resolution;
uniform float u_time;

vec2 brickTile(vec2 _st, float _zoom){
    _st *= _zoom;

    // Here is where the offset is happening
    _st.x += step(1., mod(_st.y,2.0)) * 0.5;

    return fract(_st);
}

float box(vec2 _st, vec2 _size){
    _size = vec2(0.5)-_size*0.5;
    vec2 uv = smoothstep(_size,_size+vec2(1e-4),_st);
    uv *= smoothstep(_size,_size+vec2(1e-4),vec2(1.0)-_st);
    return uv.x*uv.y;
}
```

```
void main(void){
    vec2 st = gl_FragCoord.xy/u_resolution.xy;
    vec3 color = vec3(0.0);

    // Apply the brick tiling
    st = brickTile(st,5.0);

    color = vec3(box(st,vec2(0.9)));


    gl_FragColor = vec4(color,1.0);
}
```

```
uniform vec2 u_resolution;

void main(){
  vec2 st = gl_FragCoord.xy/u_resolution.xy;
  st.x *= u_resolution.x/u_resolution.y;
  vec3 color = vec3(0.0);
  float d = 0.0;

  // Remap the space to -1. to 1.
  st = st *2.-1.;

  // Make the distance field
  d = length( abs(st)-.3 );
  // d = length( min(abs(st)-.3,0.) );
  // d = length( max(abs(st)-.3,0.) );

  // Visualize the distance field
  gl_FragColor = vec4(vec3(fract(d*10.0)),1.0);

  // Drawing with the distance field
  // gl_FragColor = vec4(vec3( step(.3,d) ),1.0);
  // gl_FragColor = vec4(vec3( step(.3,d) * step(d,.4)),1.0);
  // gl_FragColor = vec4(vec3( smoothstep(.3,.4,d)*
smoothstep(.6,.5,d)) ,1.0);
}
```



https://www.shadertoy.com/user/Danguafer

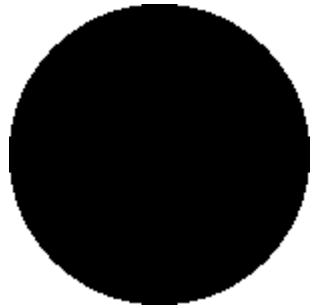


Что хотим получить?

строящийся на GPU параметрически настраиваемый спиннер

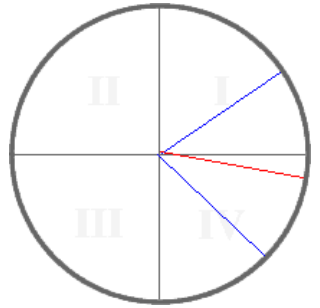
Фрагментный шейдер, описывающий круг

```
uniform vec2 u_resolution;


float PI = 3.14159;
vec2 center = u_resolution * 0.5;
float radius = min(u_resolution.x, u_resolution.y) * .5;

float circle(vec2 coord, vec2 center, float radius) {
    float distanceToCenter = distance(coord, center);
    return step(distanceToCenter, radius);
}

void main() {
  vec2 coord = vec2(gl_FragCoord);
  float isFilled = circle(coord, center, radius);
  gl_FragColor = vec4(1. - isFilled);
}
```



На окружности отсекаем сектор задавая начальный и конечный угол

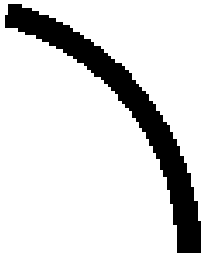
```
bool isAngleBetween(float target, float angle1, float angle2) {
  float startAngle = min(angle1, angle2);
  float endAngle = max(angle1, angle2);

  if (endAngle - startAngle < 0.1) {
    return false;
  }

  target = mod((360. + (mod(target, 360.))), 360.);
  startAngle = mod((3600000. + startAngle), 360.);
  endAngle = mod((3600000. + endAngle), 360.);

  if (startAngle < endAngle) return startAngle <= target && target <= endAngle;
  return startAngle <= target || target <= endAngle;
}
```



```
float sector(vec2 coord, vec2 center, float startAngle, float endAngle) {
  vec2 uvToCenter = coord - center;
  float angle = degrees(atan(uvToCenter.y, uvToCenter.x));
  if (isAngleBetween(angle, startAngle, endAngle)) {
    return 1.0;
  } else {
    return 0.;
  }
}

void main() {
  vec2 coord = vec2(gl_FragCoord);
  float isFilled = circle(coord, center, radius) * sector(coord, center, 0., 75.);
  gl_FragColor = vec4(1. - isFilled);
}
```

Отсекаем внутренний радиус

```
float arc(vec2 uv, vec2 center, float startAngle, float endAngle, float innerRadius, float outerRadius) {
  float result = 0.0;
  result = sector(uv, center, startAngle, endAngle) * circle(uv, center, outerRadius) * (1.0 - circle(uv, center, innerRadius));
  return result;
}

void main() {
  vec2 coord = vec2(gl_FragCoord);
  float width = 7.;
  float outerRadius = min(u_resolution.x, u_resolution.y) * .5;
  float innerRadius = outerRadius - width;
  float isFilled = arc(coord, center, 0., 75., innerRadius, outerRadius);
  gl_FragColor = vec4(1. - isFilled);
}
```



Заменим step на smoothstep чтобы получить эффект сглаживания краев

```
float circle(vec2 coord, vec2 center, float radius) {
  float distanceToCenter = distance(coord, center);
//Before
//return step(distanceToCenter, radius);
//After
  return smoothstep(distanceToCenter - 2., distanceToCenter, radius);
}
```

Анимация спиннера

Анимацию спиннера можно декомпозировать на две части:

- движение краев арки
- вращение всего спиннера

Движение краев арки

- Передний край разгоняется на протяжении всего оборота.
- В момент когда передний край завершает оборот, задний край начинает вращение сразу с максимальной скоростью, замедляясь к окончанию своего оборота.

Ускорение и замедление можно реализовать использованием функции синуса в пределе от -Pi/2 до Pi/2 передавая в виде X текущее время, точнее остаток от его деления на Pi, отняв от него Pi/2. Таким образом время всегда будет в нужном промежутке от -Pi/2 до Pi/2.

```
void main() {
  vec2 coord = vec2(gl_FragCoord);
  float width = 7.;
  float halfPI = PI * .5;
  float periodicTime = mod(u_time, PI) - halfPI;

  float outerRadius = min(u_resolution.x, u_resolution.y) * .5;
  float innerRadius = outerRadius - width;

  float startX = clamp(periodicTime, -halfPI, 0.);
  float endX = clamp(periodicTime, 0., halfPI);

  float angleVariation = sin(startX) + 1.;
  float endAngleVariation = sin(endX);

  float startAngle = 360. * angleVariation;
  float endAngle = 360. * endAngleVariation;

  float isFilled = arc(coord, center, - startAngle, - endAngle,
innerRadius, outerRadius);
  gl_FragColor = vec4(1. - isFilled);
}
```

Время храним в переменной periodicTime.

Для переднего края нас будет интересовать синус на промежутке от -Pi/2 до 0,
а для заднего края от 0 до Pi/2.

Реализуем подобное отсечение periodicTime с помощью GLSL функции clamp (переменные startX, endX).

Для отрицательных величин синуса добавим 1 и переведем startX/endX в градусы умножив на 360.

Добавим вращение. Оно ускоряется в начале периода и тормозит в конце, что соответствует графику синуса на периоде от -Pi/2 до Pi/2, однако с в два раза меньшей периодичностью.

Так же перевернем спиннер на 90 градусов изменив формулу startAngle/endAngle что бы вращение начиналось в верхней части.

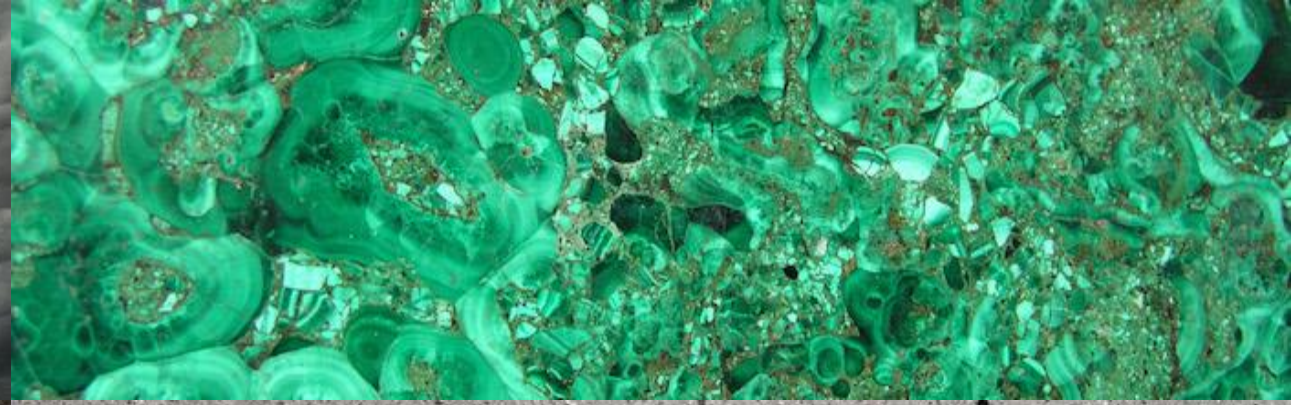
```
//main
float rotation = 180. * (sin(periodicTime) + 1.);

float startAngle = 360. * angleVariation + rotation - 90.;
float endAngle = 360. * endAngleVariation + rotation - 90.;

float isFilled = arc(coord, center, - startAngle, - endAngle, innerRadius, outerRadius);
gl_FragColor = vec4(1. - isFilled);
```

Осталось задать цвет.

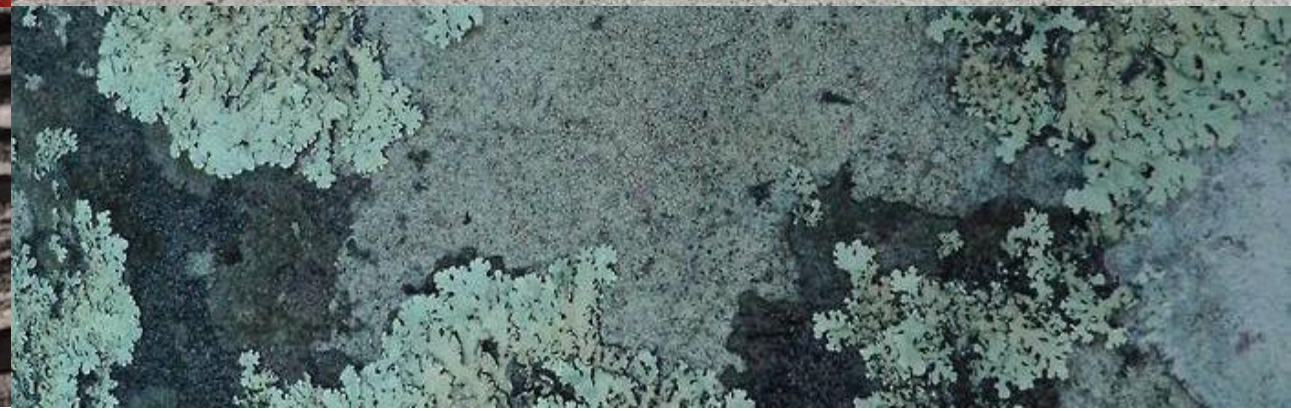
```
float width = 2.;
vec4 color = rgb(45., 121., 184.);
vec4 backgroundColor = vec4(1.);
float innerRadius = outerRadius - width;
float isFilled = arc(coord, center, - startAngle, - endAngle, innerRadius, outerRadius);
gl_FragColor = (backgroundColor - (backgroundColor - color) * isFilled);
```

```
float distanceToMouse = distance(u_mouse, gl_FragCoord.xy) * 0.085;

float width = 50. - clamp(5. * distanceToMouse, 5., 45.);

vec4 color = rgb(255. * (sin(periodicTime) + 1.), 60. * distanceToMouse / 2., 160.) * (radius / distance(gl_FragCoord.xy, center));

vec4 backgroundColor = vec4(0.);

float innerRadius = outerRadius - width;
float isFilled = arc(coord, center, - startAngle, - endAngle, innerRadius, outerRadius);
gl_FragColor = (backgroundColor - (backgroundColor - color) * isFilled);
```

Добавим зависимость цвета от угла вращения и зависимость толщины спиннера от расстояния до мыши.





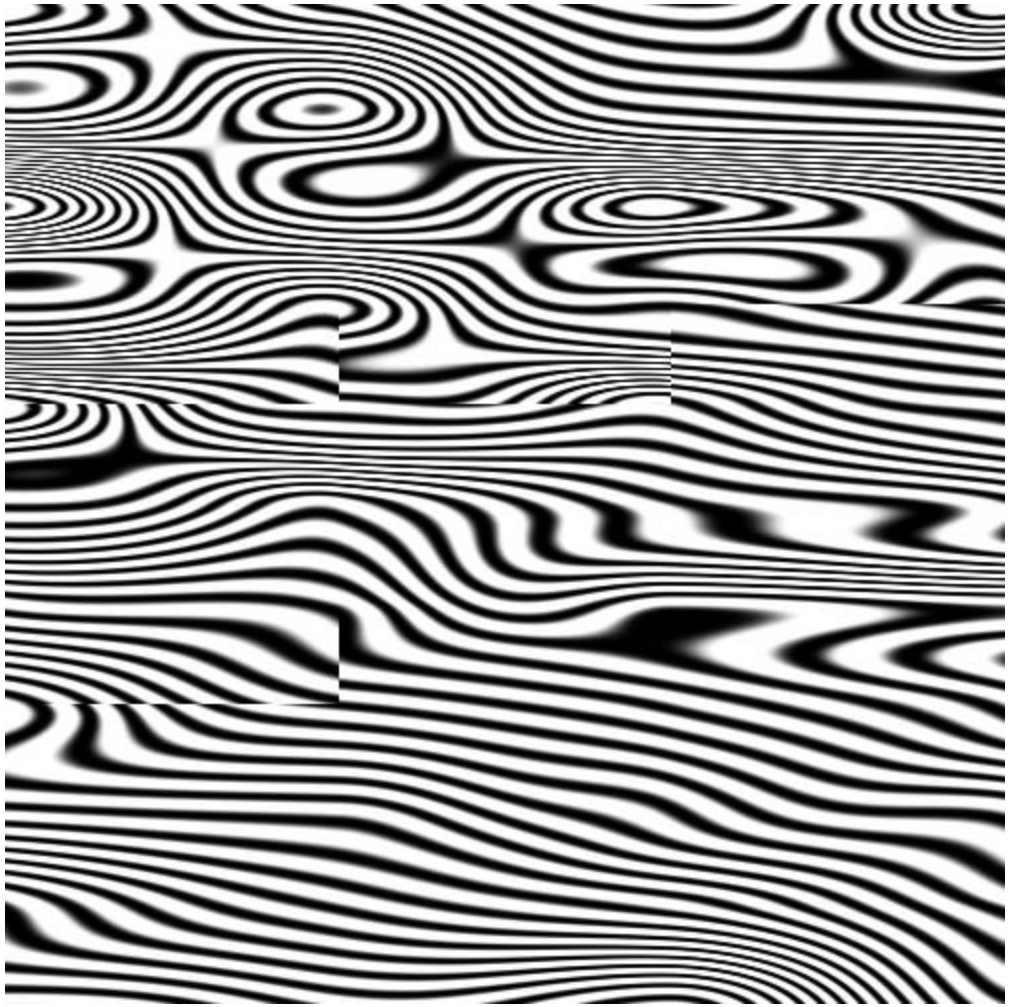
Шум значений

```
float random2(vec2 st){
    st = vec2( dot(st,vec2(127.1,311.7)), dot(st,vec2(269.5,183.3)) );
    return -1.0 + 2.0 * fract( sin( dot( st.xy, vec2(12.9898,78.233) ) ) * 43758.5453123);
}

float noise(vec2 st) {
    vec2 i = floor(st);
    vec2 f = fract(st);
    vec2 u = f*f*(3.0-2.0*f);
    return mix( mix( random2( i + vec2(0.0,0.0) ), random2( i + vec2(1.0,0.0) ), u.x),
                mix( random2( i + vec2(0.0,1.0) ), random2( i + vec2(1.0,1.0) ), u.x), u.y);
}

void main() {
    vec2 st = gl_FragCoord.xy/u_resolution.xy;
    st.x *= u_resolution.x/u_resolution.y;
    vec3 color = vec3(0.0);
    vec2 pos = vec2(st*10.0);
    color = vec3( noise(pos)*.5+.5 );
    gl_FragColor = vec4(color,1.0);
}
```

Градиентный шум

```
vec2 random2(vec2 st){
    st = vec2( dot(st,vec2(127.1,311.7)), dot(st,vec2(269.5,183.3)) );
    return -1.0 + 2.0*fract(sin(st)*43758.5453123);
}

float noise(vec2 st) {
    vec2 i = floor(st);
    vec2 f = fract(st);
    vec2 u = f*f*(3.0-2.0*f);
    return mix( mix( dot( random2(i + vec2(0.0,0.0) ), f - vec2(0.0,0.0) ), dot( random2(i + vec2(1.0,0.0) ), f - vec2(1.0,0.0) ), u.x),
                mix( dot( random2(i + vec2(0.0,1.0) ), f - vec2(0.0,1.0) ), dot( random2(i + vec2(1.0,1.0) ), f - vec2(1.0,1.0) ), u.x), u.y);
}

void main() {
    vec2 st = gl_FragCoord.xy/u_resolution.xy;
    st.x *= u_resolution.x/u_resolution.y;
    vec3 color = vec3(0.0);
    vec2 pos = vec2(st*10.0);
    color = vec3( noise(pos)*.5+.5 );
    gl_FragColor = vec4(color,1.0);
}
```

Текстура древесины

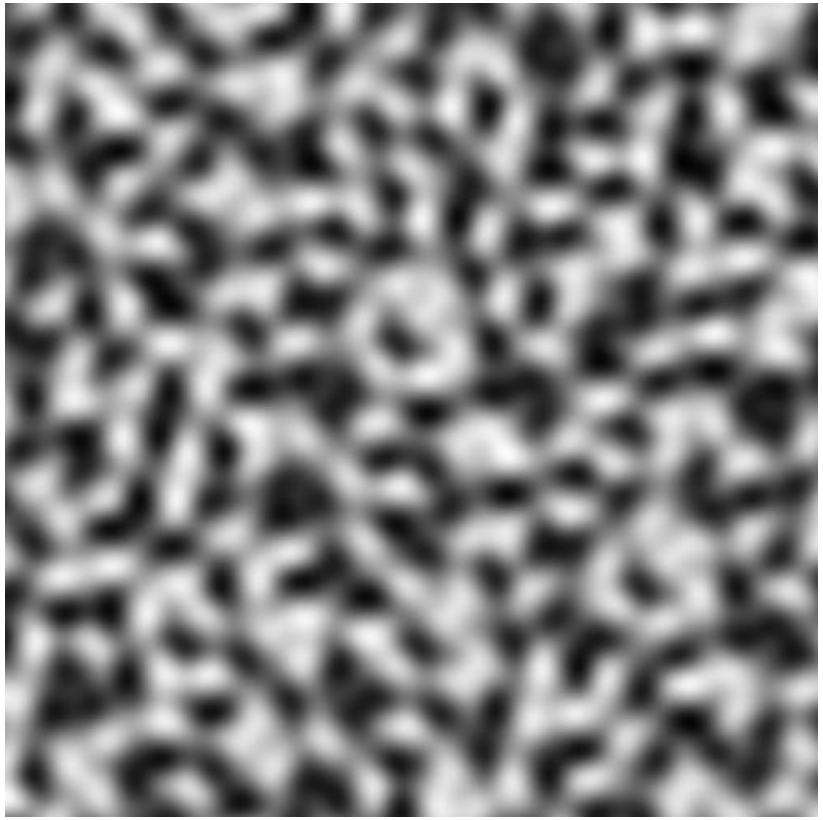


```
float random (in vec2 st) {
    return fract(sin(dot(st.xy, vec2(12.9898,78.233))) * 43758.5453123);
}

float noise(vec2 st) {
    vec2 i = floor(st);
    vec2 f = fract(st);
    vec2 u = f*f*(3.0-2.0*f);
    return mix( mix( random( i + vec2(0.0,0.0) ), random( i + vec2(1.0,0.0) ), u.x),
                mix( random( i + vec2(0.0,1.0) ), random( i + vec2(1.0,1.0) ), u.x), u.y);
}
```

```
mat2 rotate2d(float angle){
    return mat2(cos(angle),-sin(angle), sin(angle),cos(angle));
}

float lines(in vec2 pos, float b){
    float scale = 10.0;
    pos *= scale;
    return smoothstep(0.0, .5+b*.5, abs((sin(pos.x*3.1415)+b*2.0))*.5);
}

void main() {
    vec2 st = gl_FragCoord.xy/u_resolution.xy;
    st.y *= u_resolution.y/u_resolution.x;
    vec2 pos = st.yx*vec2(10.,3.);
    float pattern = pos.x;
    // Add noise
    pos = rotate2d( noise(pos) ) * pos;
    // Draw lines
    pattern = lines(pos,.5);
    gl_FragColor = vec4(vec3(pattern),1.0);
}
```

Симплексный шум

https://thebookofshaders.com/edit.php#11/2d-snoise-clear.frag



Шейдер, создающий иллюзию потока

https://thebookofshaders.com/edit.php#11/lava-lamp.frag