Вычисления на видеокартах

Лекция 10: SDF

- Signed Distance Functions
- Ray marching
- Применения в физических симуляциях, глобальном освещении и навигации роботов

Симиютин Борис
simiyutin

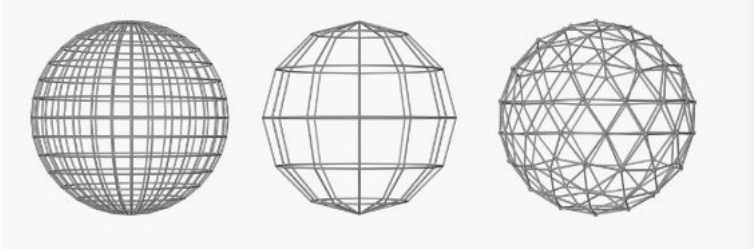
картинка:: EvilRyu

Мотивация

- Обычный способ рендеринга: рисуем модель как множество треугольничков
- Что если хотим отрисовать сферу/другой геометрический примитив?
  - В математическом смысле элементарный объект
  - Если рисовать треугольниками, то надо получить модель
- Что если хотим динамически накинуть морфологических/булевых операций? Диляция/эрозия, объединить/вычесть модели
- Что если хотим объединять сущности разных типов? Математические + полигональные?

```
// pseudocode
bool sphere(vec3 center, float r, vec3 pos) {
    return length(pos - center) == r;
}
```

VS

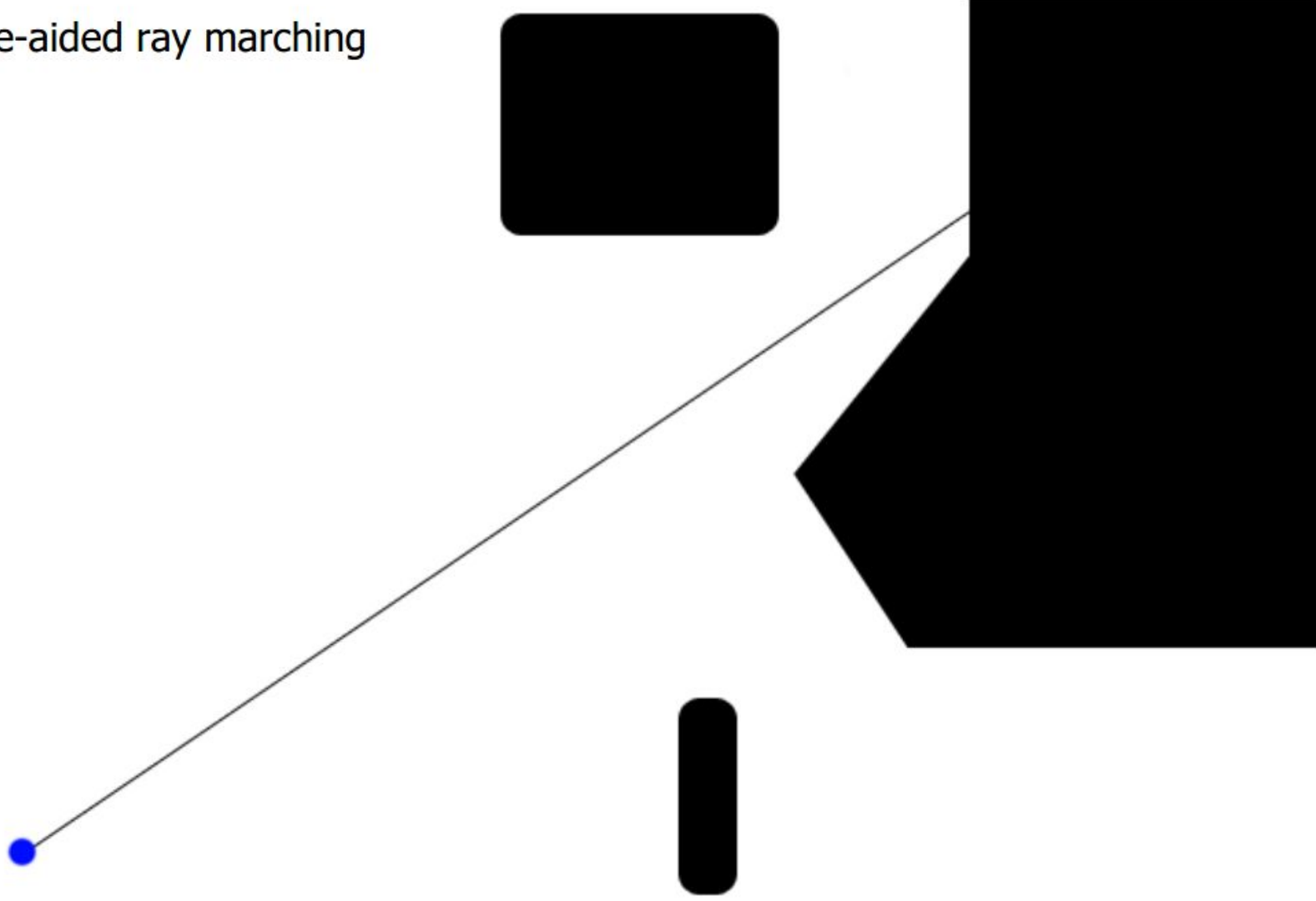


Signed Distance Field

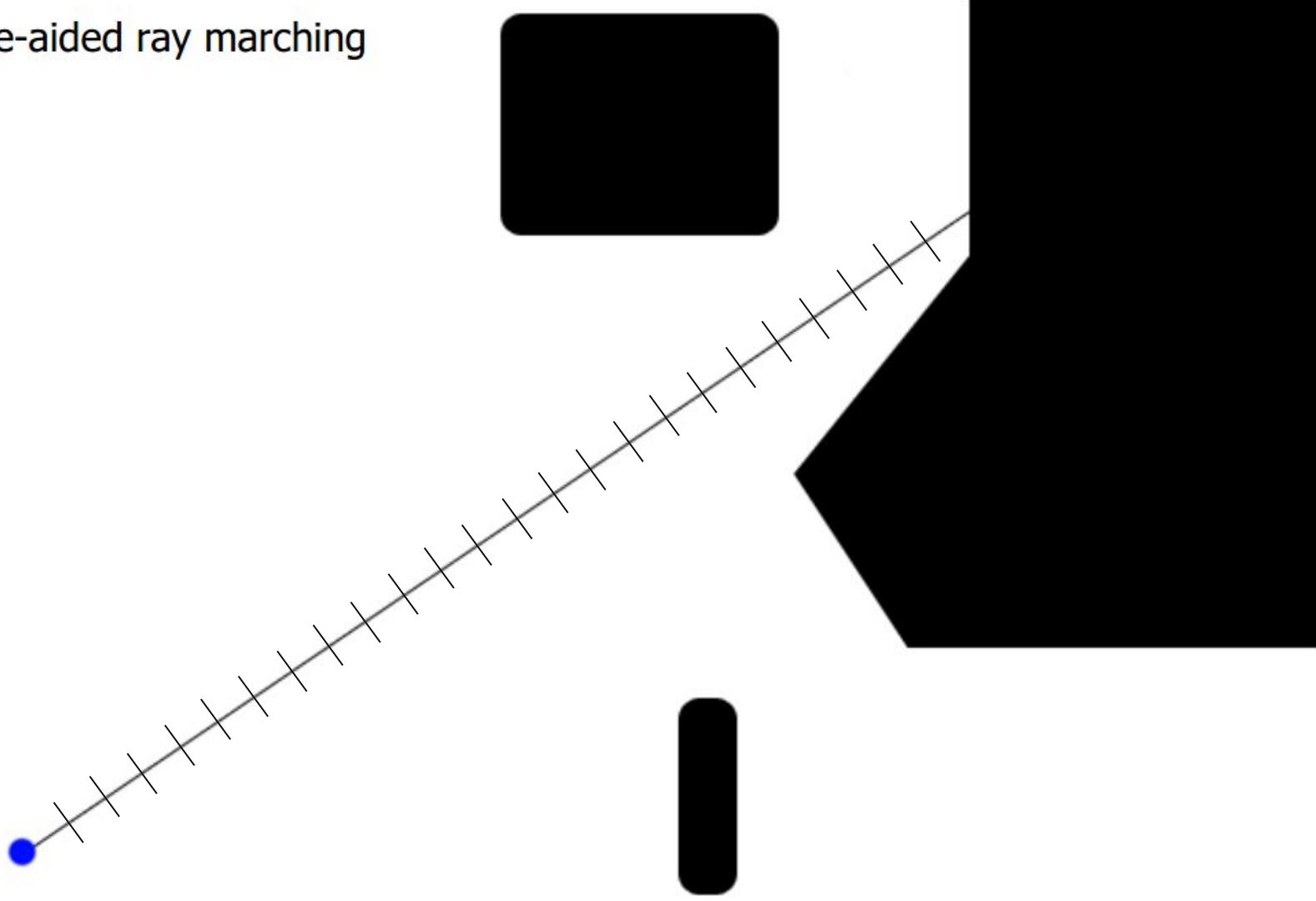
- Функция, возвращающая для заданной точки расстояние до поверхности объекта
- Расстояние положительное если точка снаружи объекта
- Расстояние отрицательное если точка внутри
- Равно нулю на границе

https://www.shadertoy.com/view/3tyBzV

Distance-aided ray marching



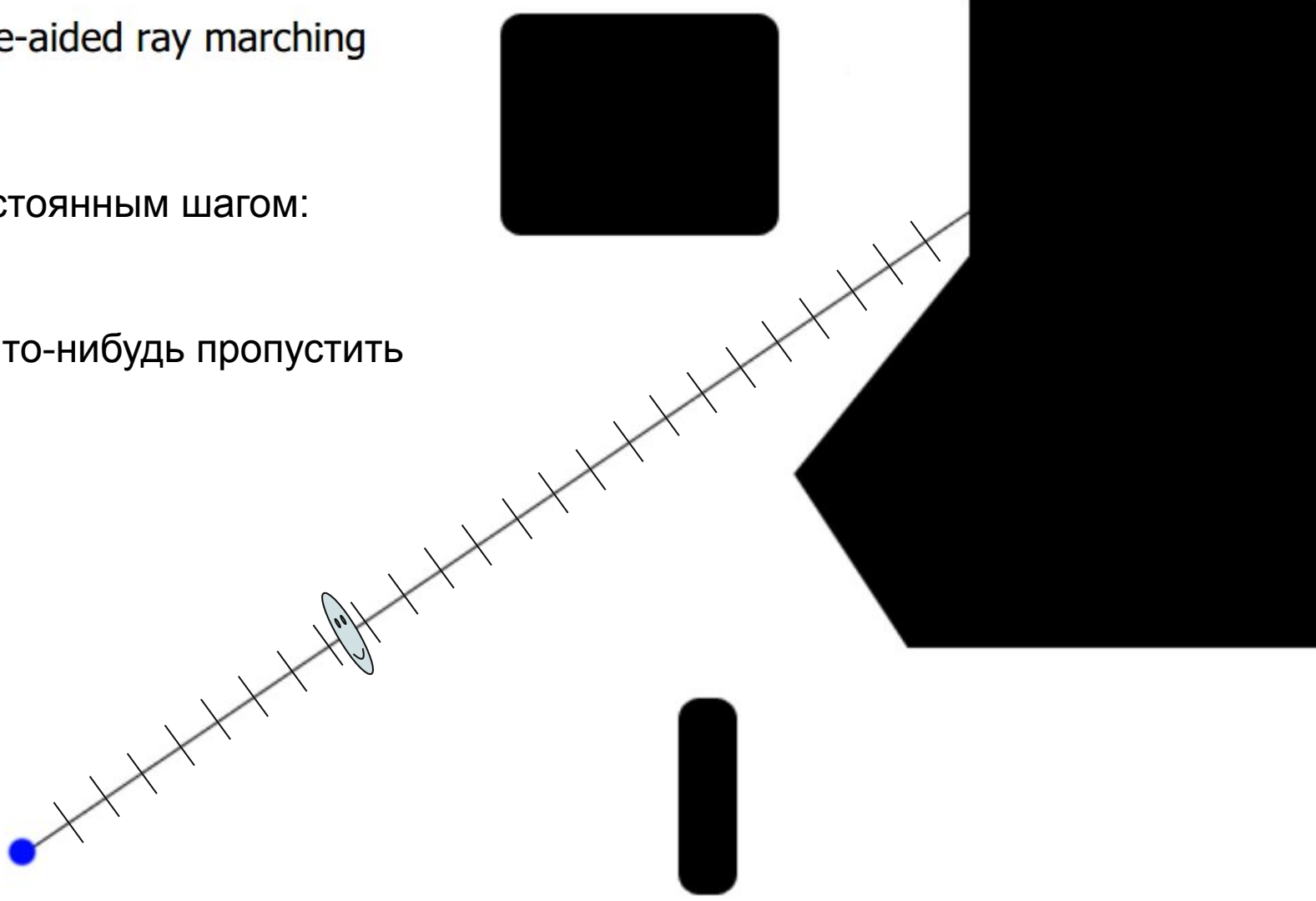
Distance-aided ray marching

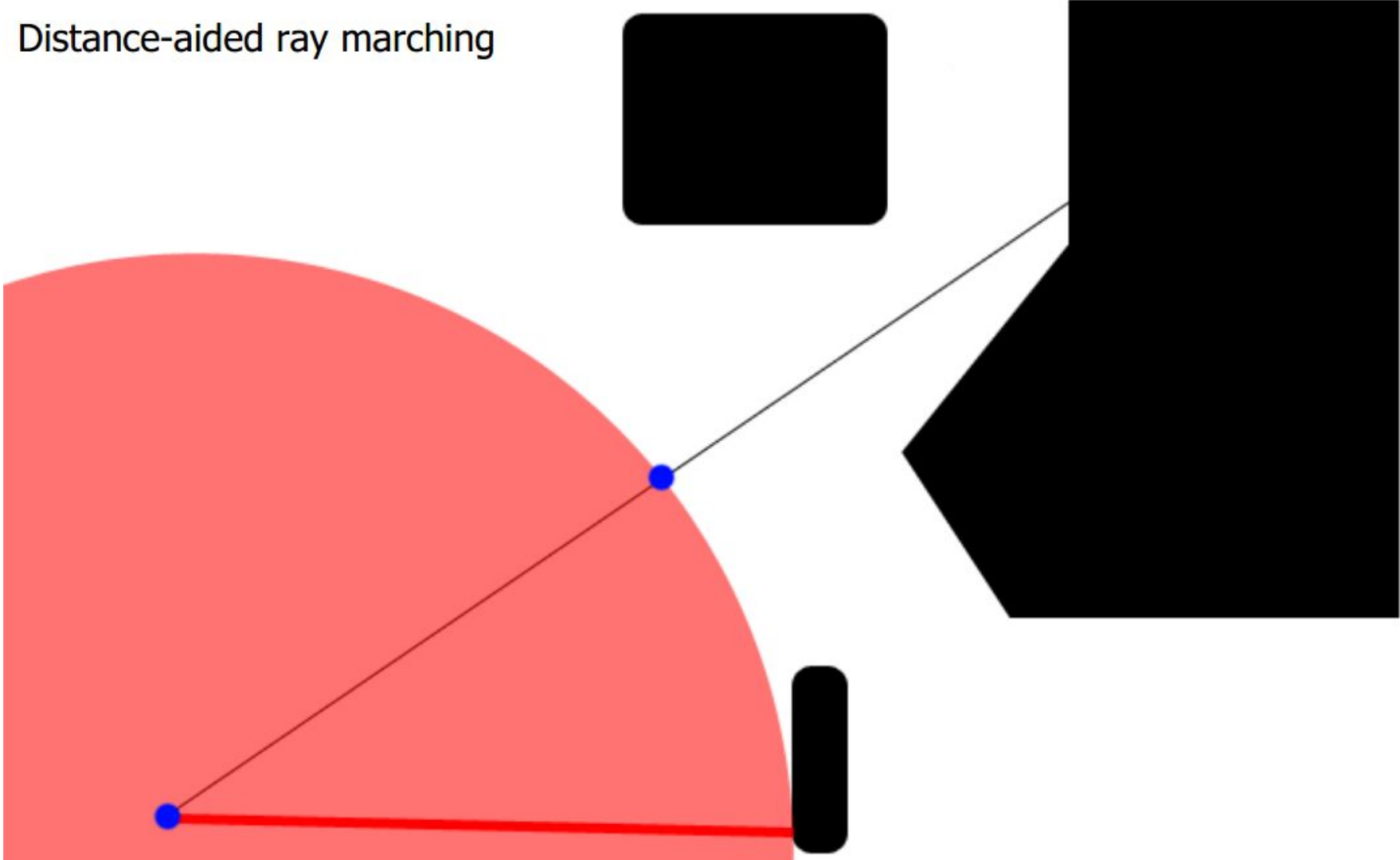


Distance-aided ray marching

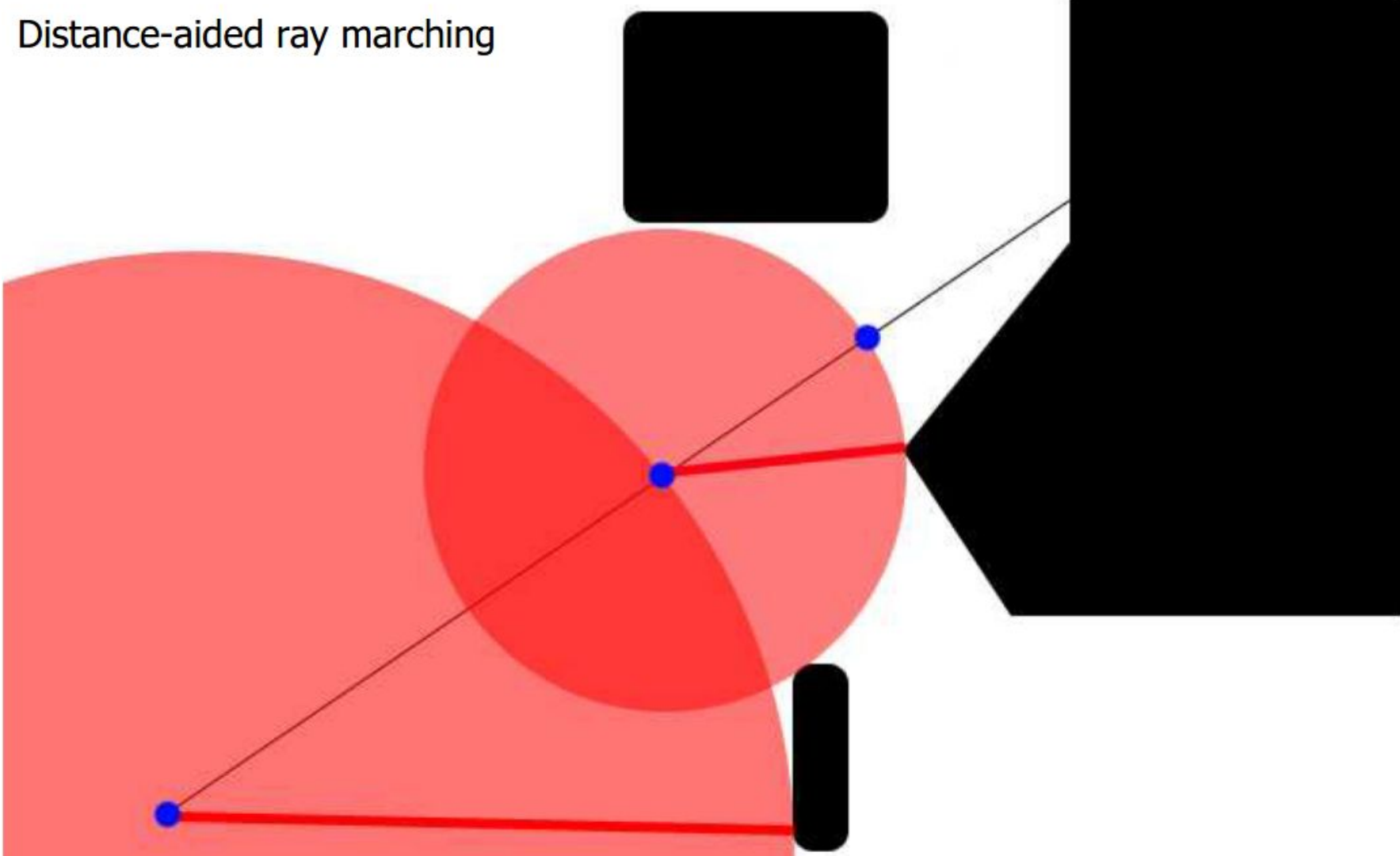
Подход с постоянным шагом:

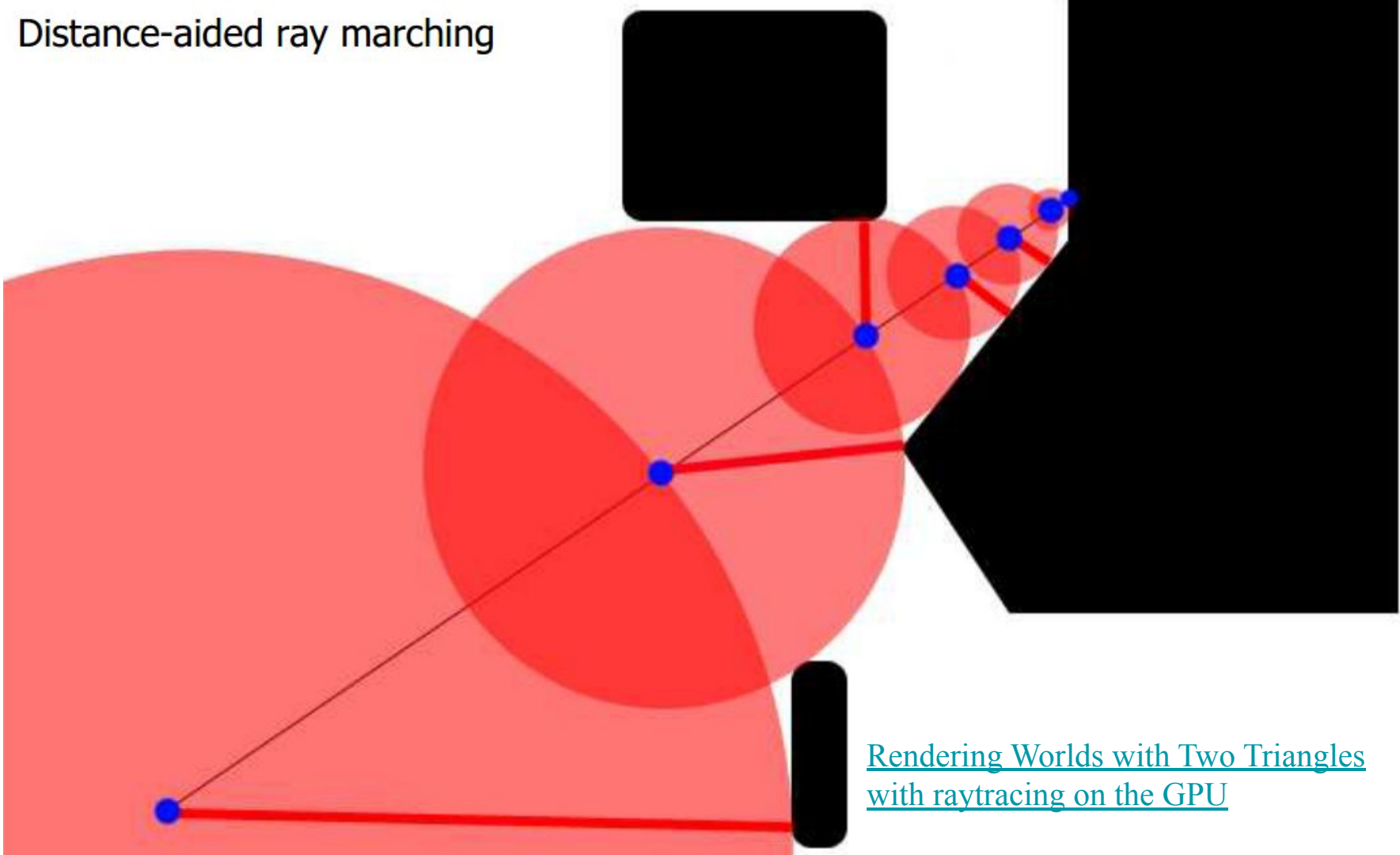
- Долго
- Можно что-нибудь пропустить




Distance-aided ray marching

Distance-aided ray marching



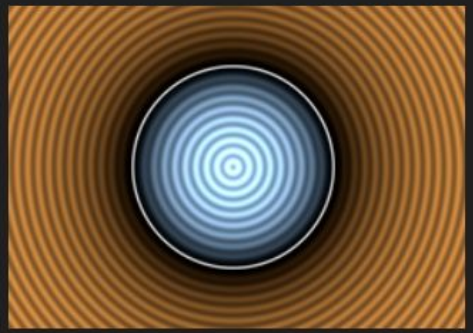


[Rendering Worlds with Two Triangles with raytracing on the GPU]

Signed Distance Field

- Альтернатива полигональной геометрии, 2D, 3D
- Позволяет выполнять трассировку лучей быстрее чем подход с константным шагом
- Большая гибкость в математическом формулировании геометрии сцены, в несколько строчек можно:
  - выполнять булевы операции над объемами
  - закручивать и загибать фигуры
  - подсчитывать освещение и затенение
  - процедурно генерировать шероховатость материала, геологический ландшафт, текстуру
- Можно использовать для ускорения трассировки полигональных моделей, векторизации дискретных симуляций (например, ткань)

Примеры SDF



Circle - exact (https://www.shadertoy.com/view/3ltSW2)

```
float sdCircle( vec2 p, float r )
{
    return length(p) - r;
}
```

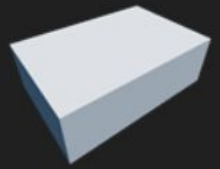
Sphere - exact (https://www.shadertoy.com/view/Xds3zN)

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

Примеры SDF

Box - exact (https://www.youtube.com/watch?v=62-pRVZuS5c)

```
float sdBox( in vec2 p, in vec2 b )
{
    vec2 d = abs(p)-b;
    return length(max(d,0.0)) + min(max(d.x,d.y),0.0);
}
```



Box - exact (Youtube Tutorial with derivation: https://www.youtube.com/watch?v=62-pRVZuS5c)

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

Примеры SDF

Hexagram - exact (https://www.shadertoy.com/view/tt23RR)

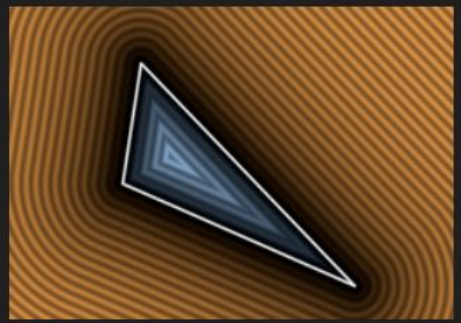
```
float sdHexagram( in vec2 p, in float r )
{
    const vec4 k = vec4(-0.5,0.8660254038,0.5773502692,1.7320508076);
    p = abs(p);
    p -= 2.0*min(dot(k.xy,p),0.0)*k.xy;
    p -= 2.0*min(dot(k.yx,p),0.0)*k.yx;
    p -= vec2(clamp(p.x,r*k.z,r*k.w),r);
    return length(p)*sign(p.y);
}
```

Примеры SDF

Heart - exact (https://www.shadertoy.com/view/3tyBzV)

```
float sdHeart( in vec2 p )
{
    p.x = abs(p.x);

    if( p.y+p.x>1.0 )
        return sqrt(dot2(p-vec2(0.25,0.75))) - sqrt(2.0)/4.0;
    return sqrt(min(dot2(p-vec2(0.00,1.00)),
                    dot2(p-0.5*max(p.x+p.y,0.0)))) * sign(p.x-p.y);
}
```

Примеры SDF



Triangle - exact (https://www.shadertoy.com/view/XsXSz4)

```
float sdTriangle( in vec2 p, in vec2 p0, in vec2 p1, in vec2 p2 )
{
    vec2 e0 = p1-p0, e1 = p2-p1, e2 = p0-p2;
    vec2 v0 = p -p0, v1 = p -p1, v2 = p -p2;
    vec2 pq0 = v0 - e0*clamp( dot(v0,e0)/dot(e0,e0), 0.0, 1.0 );
    vec2 pq1 = v1 - e1*clamp( dot(v1,e1)/dot(e1,e1), 0.0, 1.0 );
    vec2 pq2 = v2 - e2*clamp( dot(v2,e2)/dot(e2,e2), 0.0, 1.0 );
    float s = sign( e0.x*e2.y - e0.y*e2.x );
    vec2 d = min(min(vec2(dot(pq0,pq0), s*(v0.x*e0.y-v0.y*e0.x)),
                     vec2(dot(pq1,pq1), s*(v1.x*e1.y-v1.y*e1.x))),
                     vec2(dot(pq2,pq2), s*(v2.x*e2.y-v2.y*e2.x)));
    return -sqrt(d.x)*sign(d.y);
}
```

Примеры SDF

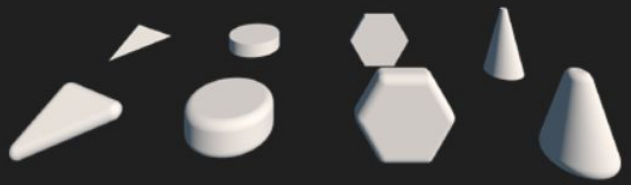
Можно получать 3D объекты из 2D путем вращения и экструзии

```
float opRevolution( in vec3 p, in sdf2d primitive, in float o )
{
    vec2 q = vec2( length(p.xz) - o, p.y );
    return primitive(q)
}
```

```
float opExtrusion( in vec3 p, in sdf2d primitive, in float h )
{
    float d = primitive(p.xy)
    vec2 w = vec2( d, abs(p.z) - h );
    return min(max(w.x,w.y),0.0) + length(max(w,0.0));
}
```

Примеры SDF

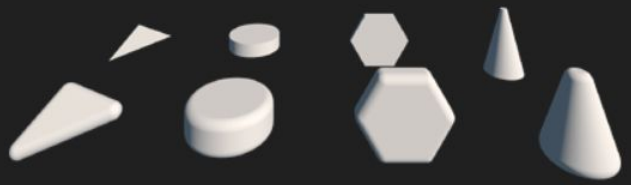
Морфология, булевы операции



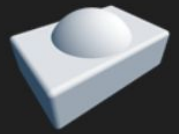
```
float opRound( in sdf3d primitive, in float rad )
{
    return primitive(p) - rad
}
```

Примеры SDF

Морфология, булевы операции



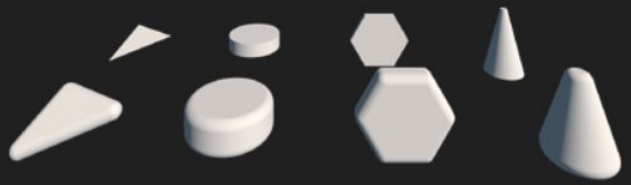
```
float opRound( in sdf3d primitive, in float rad )
{
    return primitive(p) - rad
}
```



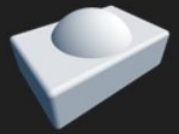
```
float opUnion( float d1, float d2 )
{
    return ?????
}
```

Примеры SDF

Морфология, булевы операции



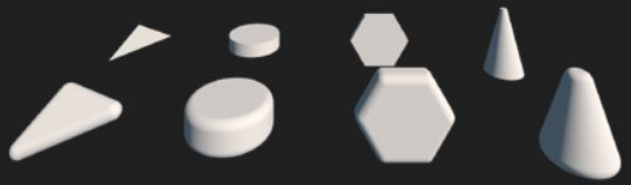
```
float opRound( in sdf3d primitive, in float rad )
{
    return primitive(p) - rad
}
```



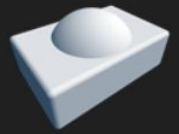
```
float opUnion( float d1, float d2 )
{
    return min(d1,d2);
}
```

Примеры SDF

Морфология, булевы операции



```
float opRound( in sdf3d primitive, in float rad )
{
    return primitive(p) - rad
}
```



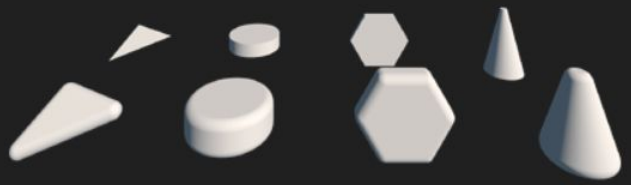
```
float opUnion( float d1, float d2 )
{
    return min(d1,d2);
}
```

```
float opIntersection( float d1, float d2 )
{
    return
}
```

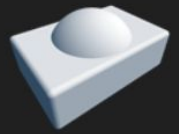
?????

Примеры SDF

Морфология, булевы операции



```
float opRound( in sdf3d primitive, in float rad )
{
    return primitive(p) - rad
}
```

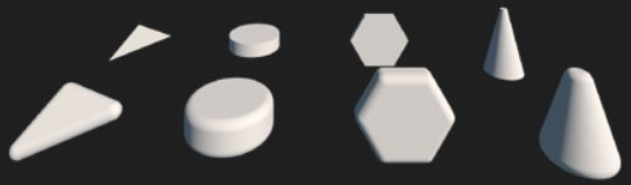


```
float opUnion( float d1, float d2 )
{
    return min(d1,d2);
}
```

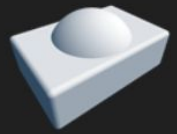
```
float opIntersection( float d1, float d2 )
{
    return max(d1,d2);
}
```

Примеры SDF

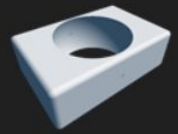
Морфология, булевы операции



```
float opRound( in sdf3d primitive, in float rad )
{
    return primitive(p) - rad
}
```



```
float opUnion( float d1, float d2 )
{
    return min(d1,d2);
}
```

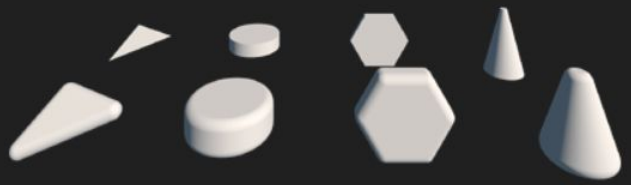


```
float opSubtraction( float d1, float d2 )
{
    return ?????;
}
```

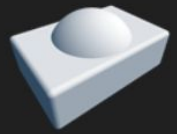
```
float opIntersection( float d1, float d2 )
{
    return max(d1,d2);
}
```

Примеры SDF

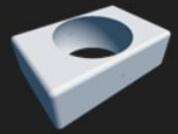
Морфология, булевы операции



```
float opRound( in sdf3d primitive, in float rad )
{
    return primitive(p) - rad
}
```



```
float opUnion( float d1, float d2 )
{
    return min(d1,d2);
}
```



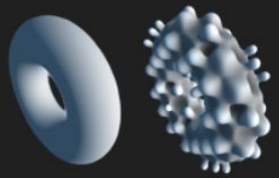
```
float opSubtraction( float d1, float d2 )
{
    return max(-d1,d2);
}
```

```
float opIntersection( float d1, float d2 )
{
    return max(d1,d2);
}
```

Примеры SDF

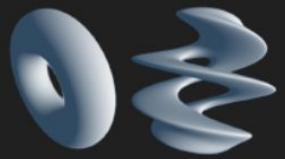
Displacement

The displacement example below is using sin(20*p.x)*sin(20*p.y)*sin(20*p.z) as displacement pattern, but you can of course use anything you might imagine.



```
float opDisplace( in sdf3d primitive, in vec3 p )
{
    float d1 = primitive(p);
    float d2 = displacement(p);
    return d1+d2;
}
```

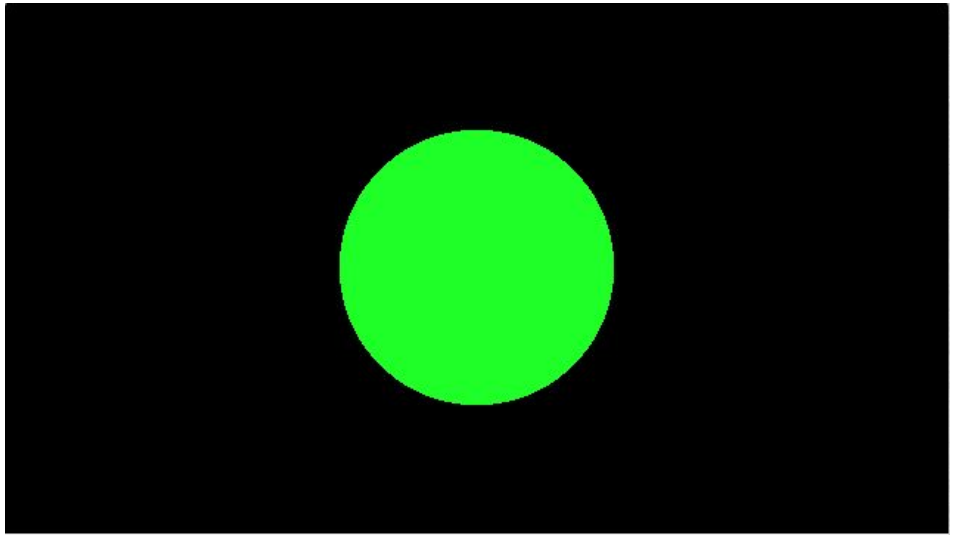
Twist



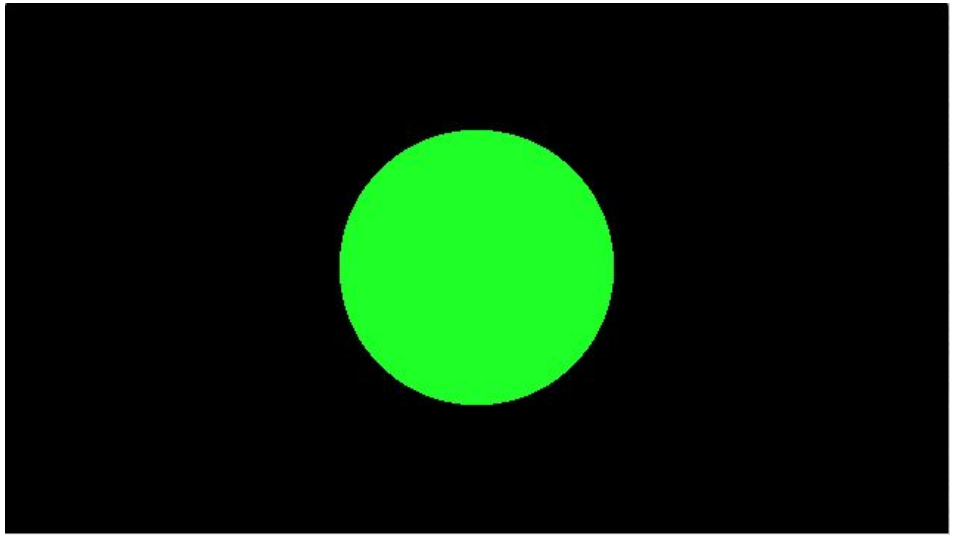
```
float opTwist( in sdf3d primitive, in vec3 p )
{
    const float k = 10.0; // or some other amount
    float c = cos(k*p.y);
    float s = sin(k*p.y);
    mat2  m = mat2(c,-s,s,c);
    vec3  q = vec3(m*p.xz,p.y);
    return primitive(q);
}
```

- Пример: Painting a Character with Maths

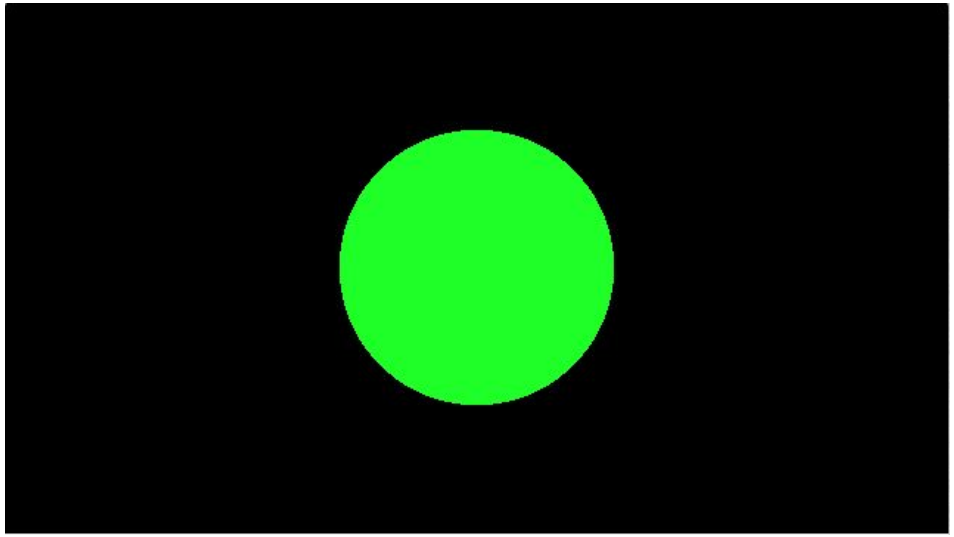
```
float sdfSphere(vec3 center, float r, vec3 pos) {
    return length(pos - center) - r;
}
```



```
float sdfSphere(vec3 center, float r, vec3 pos) {
    return length(pos - center) - r;
}
vec4 sdfTotal(vec3 pos) {
    vec4 res = vec4(INF, BLACK);
    // sphere
    {
        float dist = sdfSphere(vec3(0, 0, 2), 0.5, pos);
        if (dist < res.x) {
            res = vec4(dist, GREEN);
        }
    }
    return res;
}
// returns depth, color
vec4 traceScene(vec3 from, vec3 dir) {
    float depth = 0.f;
    for (int iter = 0; iter < 1000; ++iter) {
        vec4 dist = sdfTotal(from + depth * dir);
        if (dist.x < EPS) {
            return vec4(depth, dist.yzw);
        }
        depth += dist.x;
    }
    return vec4(INF, BLACK);
}
```

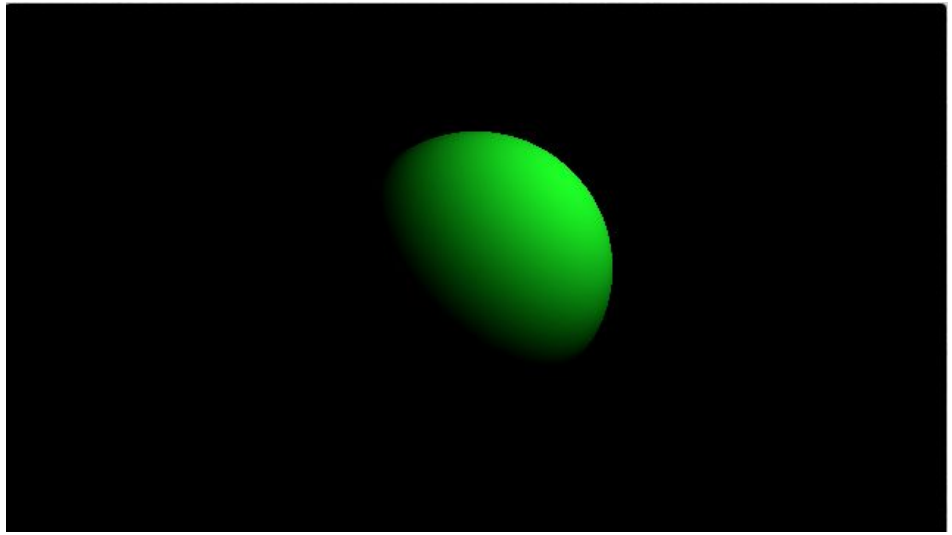


```glsl
float sdfSphere(vec3 center, float r, vec3 pos) {
    return length(pos - center) - r;
}
vec4 sdfTotal(vec3 pos) {
    vec4 res = vec4(INF, BLACK);
    // sphere
    {
        float dist = sdfSphere(vec3(0, 0, 2), 0.5, pos);
        if (dist < res.x) {
            res = vec4(dist, GREEN);
        }
    }
    return res;
}
// returns depth, color
vec4 traceScene(vec3 from, vec3 dir) {
    float depth = 0.f;
    for (int iter = 0; iter < 1000; ++iter) {
        vec4 dist = sdfTotal(from + depth * dir);
        if (dist.x < EPS) {
            return vec4(depth, dist.yzw);
        }
        depth += dist.x;
    }
    return vec4(INF, BLACK);
}
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Normalized pixel coordinates (from 0 to 1)
    vec2 uv = fragCoord/iResolution.y;
    float aspect = iResolution.x / iResolution.y;

    vec3 from = vec3(0, 0, 0);
    vec3 dir = normalize(vec3(uv - 0.5 * vec2(aspect, 1), 1.0));
    vec4 p = traceScene(from, dir);

    vec3 pos = from + p.x * dir;
    vec3 col = p.yzw;

    // Output to screen
    fragColor = vec4(col,1.0);
}
```



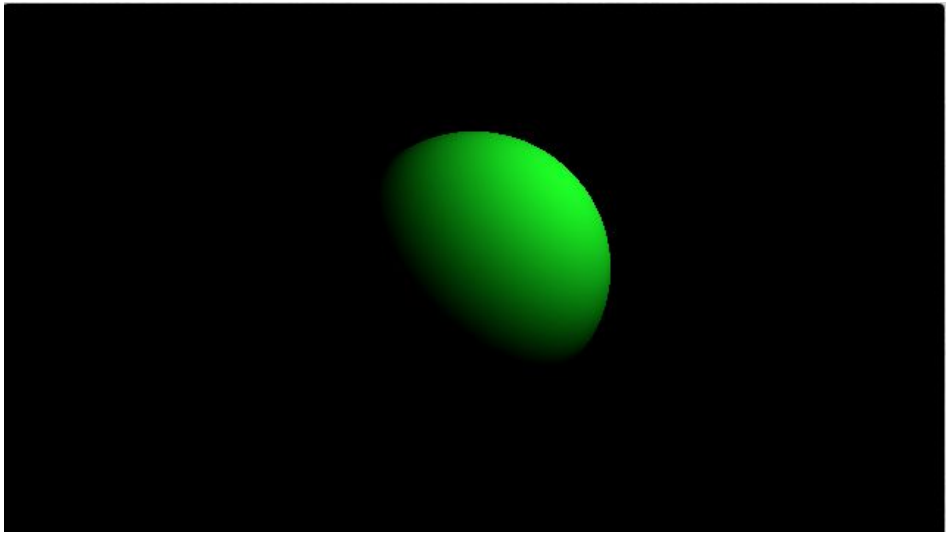
Как найти нормаль?

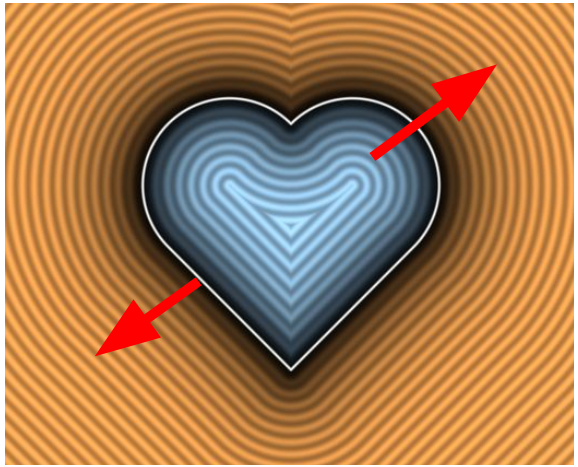
```
float shading(vec3 pos, vec3 light_pos) {
    vec3 n = normal(pos);
    vec3 dir_to_light = normalize(light_pos - pos);
    return max(0.f, dot(n, dir_to_light));
}
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Normalized pixel coordinates (from 0 to 1)
    vec2 uv = fragCoord/iResolution.y;
    float aspect = iResolution.x / iResolution.y;

    vec3 from = vec3(0, 0, 0);
    vec3 dir = normalize(vec3(uv - 0.5 * vec2(aspect, 1), 1.0));
    vec4 p = traceScene(from, dir);

    vec3 pos = from + p.x * dir;
    vec3 col = p.yzw;

    vec3 light_pos = vec3(1.f + sin(iTime), 1.f, 1.f);
    col *= shading(pos, light_pos);

    // Output to screen
    fragColor = vec4(col,1.0);
}
```



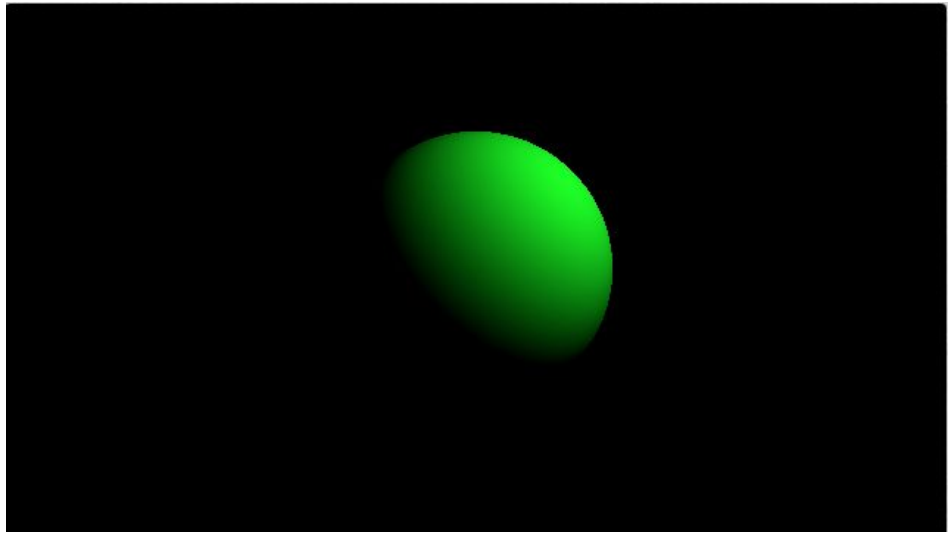
Как найти нормаль?

```
float shading(vec3 pos, vec3 light_pos) {
    vec3 n = normal(pos);
    vec3 dir_to_light = normalize(light_pos - pos);
    return max(0.f, dot(n, dir_to_light));
}
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Normalized pixel coordinates (from 0 to 1)
    vec2 uv = fragCoord/iResolution.y;
    float aspect = iResolution.x / iResolution.y;

    vec3 from = vec3(0, 0, 0);
    vec3 dir = normalize(vec3(uv - 0.5 * vec2(aspect, 1), 1.0));
    vec4 p = traceScene(from, dir);

    vec3 pos = from + p.x * dir;
    vec3 col = p.yzw;

    vec3 light_pos = vec3(1.f + sin(iTime), 1.f, 1.f);
    col *= shading(pos, light_pos);

    // Output to screen
    fragColor = vec4(col,1.0);
}
```

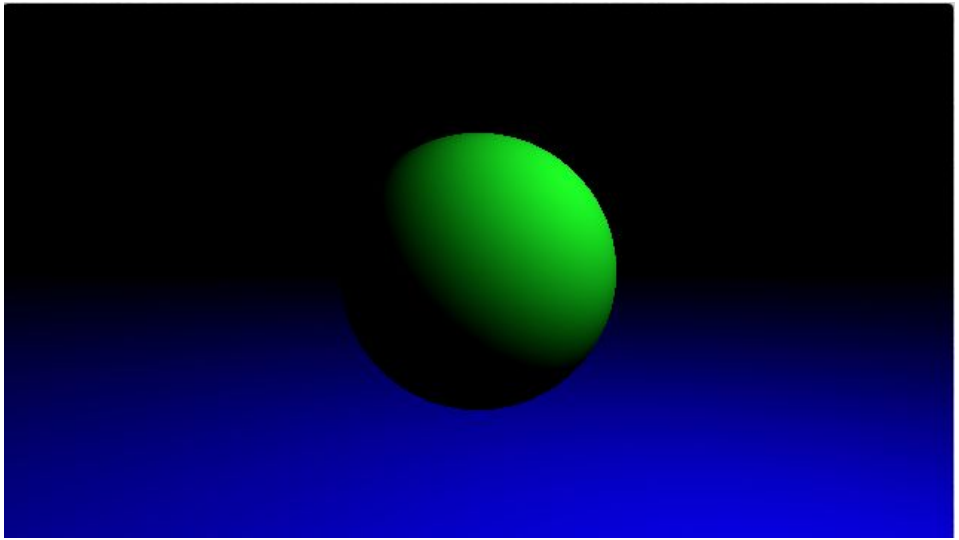




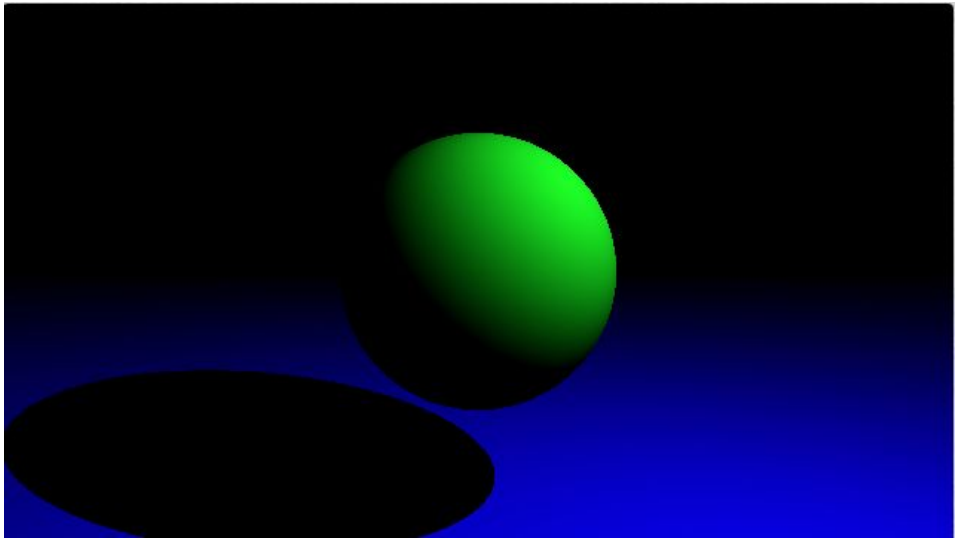
```
vec3 gradient(vec3 pos) {
    float dfdx = (sdfTotal(pos + vec3(EPS, 0, 0)).x
                - sdfTotal(pos - vec3(EPS, 0, 0)).x) / (2.f * EPS);
    float dfdy = (sdfTotal(pos + vec3(0, EPS, 0)).x
                - sdfTotal(pos - vec3(0, EPS, 0)).x) / (2.f * EPS);
    float dfdz = (sdfTotal(pos + vec3(0, 0, EPS)).x
                - sdfTotal(pos - vec3(0, 0, EPS)).x) / (2.f * EPS);
    return vec3(dfdx, dfdy, dfdz);
}
vec3 normal(vec3 pos) {
    return normalize(gradient(pos));
}
float shading(vec3 pos, vec3 light_pos) {
    vec3 n = normal(pos);
    vec3 dir_to_light = normalize(light_pos - pos);
    return max(0.f, dot(n, dir_to_light));
}
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Normalized pixel coordinates (from 0 to 1)
    vec2 uv = fragCoord/iResolution.y;
    float aspect = iResolution.x / iResolution.y;

    vec3 from = vec3(0, 0, 0);
    vec3 dir = normalize(vec3(uv - 0.5 * vec2(aspect, 1), 1.0));
    vec4 p = traceScene(from, dir);

    vec3 pos = from + p.x * dir;
    vec3 col = p.yzw;

    vec3 light_pos = vec3(1.f + sin(iTime), 1.f, 1.f);
    col *= shading(pos, light_pos);

    // Output to screen
    fragColor = vec4(col,1.0);
}
```



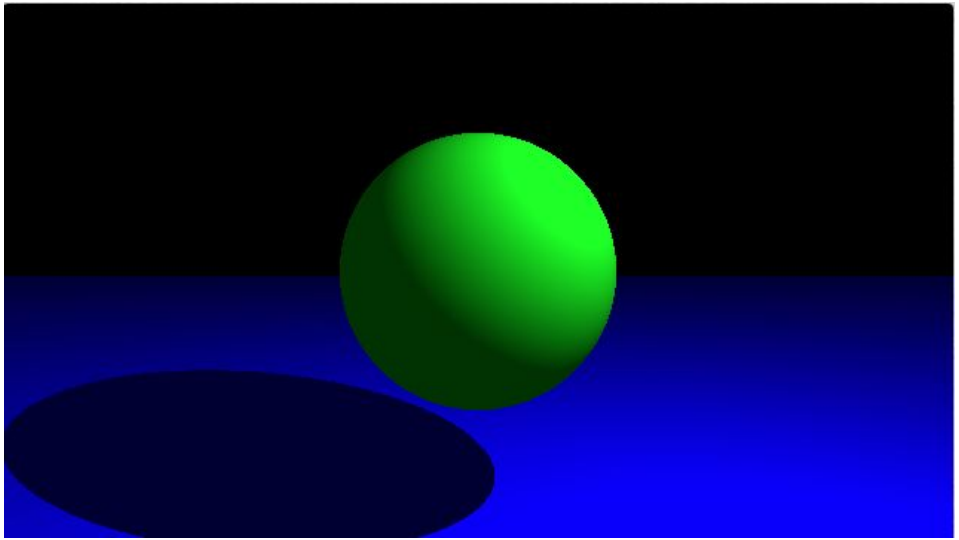
```
vec3 gradient(vec3 pos) {
    float dfdx = (sdfTotal(pos + vec3(EPS, 0, 0)).x
                - sdfTotal(pos - vec3(EPS, 0, 0)).x) / (2.f * EPS);
    float dfdy = (sdfTotal(pos + vec3(0, EPS, 0)).x
                - sdfTotal(pos - vec3(0, EPS, 0)).x) / (2.f * EPS);
    float dfdz = (sdfTotal(pos + vec3(0, 0, EPS)).x
                - sdfTotal(pos - vec3(0, 0, EPS)).x) / (2.f * EPS);
    return vec3(dfdx, dfdy, dfdz);
}
vec3 normal(vec3 pos) {
    return normalize(gradient(pos));
}
float shading(vec3 pos, vec3 light_pos) {
    vec3 n = normal(pos);
    vec3 dir_to_light = normalize(light_pos - pos);
    return max(0.f, dot(n, dir_to_light));
}
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Normalized pixel coordinates (from 0 to 1)
    vec2 uv = fragCoord/iResolution.y;
    float aspect = iResolution.x / iResolution.y;

    vec3 from = vec3(0, 0, 0);
    vec3 dir = normalize(vec3(uv - 0.5 * vec2(aspect, 1), 1.0));
    vec4 p = traceScene(from, dir);

    vec3 pos = from + p.x * dir;
    vec3 col = p.yzw;

    vec3 light_pos = vec3(1.f + sin(iTime), 1.f, 1.f);
    col *= shading(pos, light_pos);

    // Output to screen
    fragColor = vec4(col,1.0);
}
```



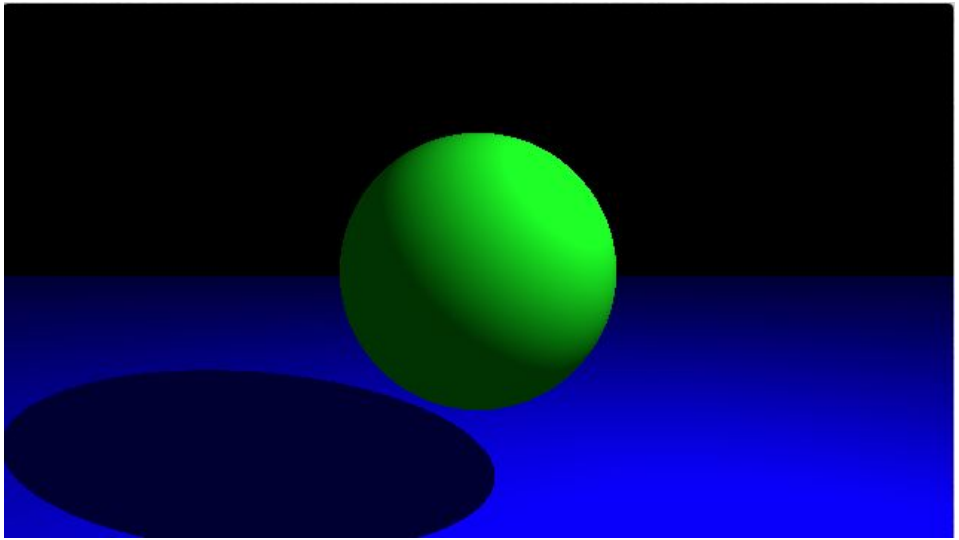
```
float castShadow(vec3 pos, vec3 light_pos) {
    float estimated = length(pos - light_pos);
    vec3 dir = normalize(light_pos - pos);
    float actual = traceScene(pos + EPS * normal(pos), dir).x;
    if (actual + 2.f * EPS < estimated)
        return 0.f;
    return 1.f;
}
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Normalized pixel coordinates (from 0 to 1)
    vec2 uv = fragCoord/iResolution.y;
    float aspect = iResolution.x / iResolution.y;

    vec3 from = vec3(0, 0, 0);
    vec3 dir = normalize(vec3(uv - 0.5 * vec2(aspect, 1), 1.0));
    vec4 p = traceScene(from, dir);

    vec3 pos = from + p.x * dir;
    vec3 col = p.yzw;

    vec3 light_pos = vec3(1.f + sin(iTime), 1.f, 1.f);
    col *= shading(pos, light_pos);
    col *= castShadow(pos, light_pos);

    // Output to screen
    fragColor = vec4(col,1.0);
}
```



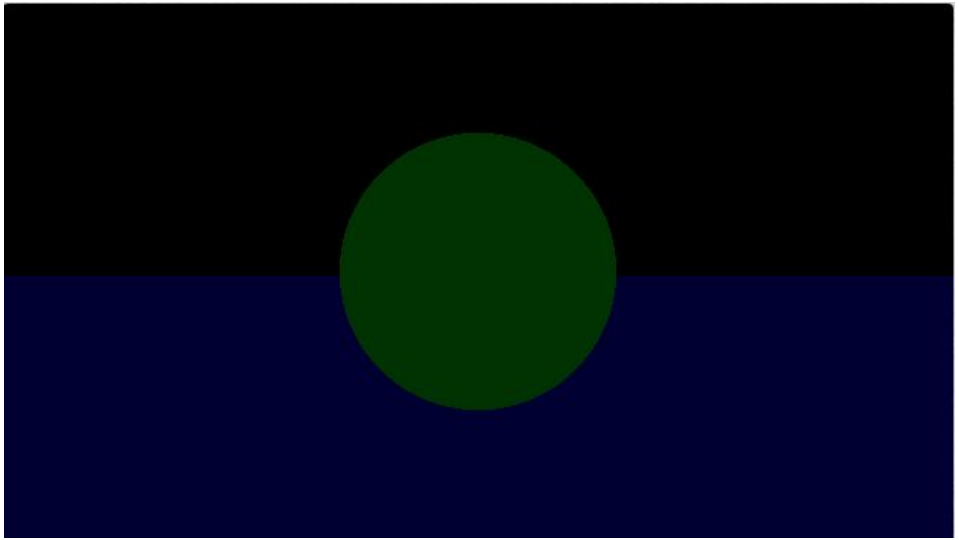
```
float castShadow(vec3 pos, vec3 light_pos) {
    float estimated = length(pos - light_pos);
    vec3 dir = normalize(light_pos - pos);
    float actual = traceScene(pos + EPS * normal(pos), dir).x;
    if (actual + 2.f * EPS < estimated)
        return 0.f;
    return 1.f;
}
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Normalized pixel coordinates (from 0 to 1)
    vec2 uv = fragCoord/iResolution.y;
    float aspect = iResolution.x / iResolution.y;

    vec3 from = vec3(0, 0, 0);
    vec3 dir = normalize(vec3(uv - 0.5 * vec2(aspect, 1), 1.0));
    vec4 p = traceScene(from, dir);

    vec3 pos = from + p.x * dir;
    vec3 col = p.yzw;

    vec3 light_pos = vec3(1.f + sin(iTime), 1.f, 1.f);
    //col *= shading(pos, light_pos);
    //col *= castShadow(pos, light_pos);
    col *= min(0.2 + shading(pos, light_pos)
                   * castShadow(pos, light_pos), 1.f);

    // Output to screen
    fragColor = vec4(col,1.0);
}
```



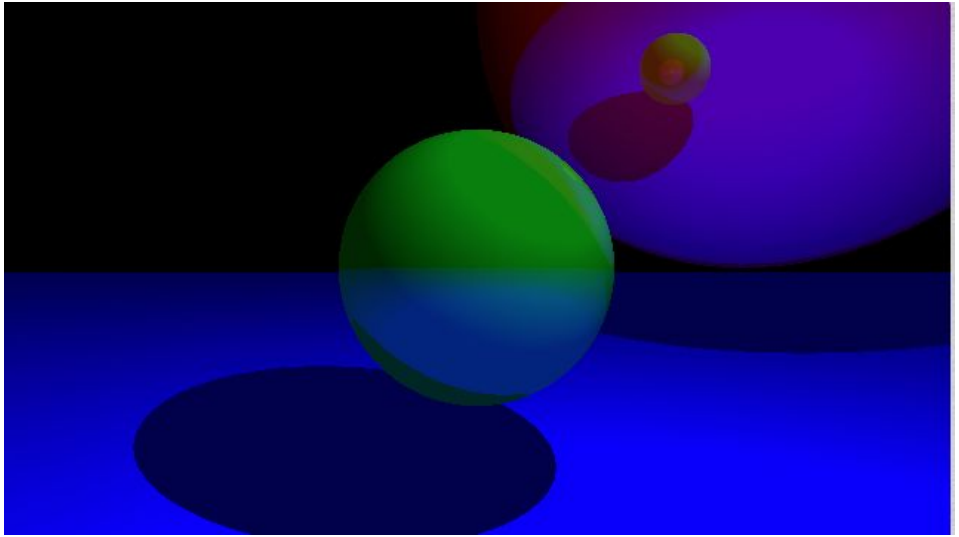
```glsl
float castShadow(vec3 pos, vec3 light_pos) {
    float estimated = length(pos - light_pos);
    vec3 dir = normalize(light_pos - pos);
    float actual = traceScene(pos + EPS * normal(pos), dir).x;
    if (actual + 2.f * EPS < estimated)
        return 0.f;
    return 1.f;
}
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Normalized pixel coordinates (from 0 to 1)
    vec2 uv = fragCoord/iResolution.y;
    float aspect = iResolution.x / iResolution.y;

    vec3 from = vec3(0, 0, 0);
    vec3 dir = normalize(vec3(uv - 0.5 * vec2(aspect, 1), 1.0));
    vec4 p = traceScene(from, dir);

    vec3 pos = from + p.x * dir;
    vec3 col = p.yzw;

    vec3 light_pos = vec3(1.f + sin(iTime), 1.f, 1.f);
    //col *= shading(pos, light_pos);
    //col *= castShadow(pos, light_pos);
    col *= min(0.2 + shading(pos, light_pos)
                   * castShadow(pos, light_pos), 1.f);

    // Output to screen
    fragColor = vec4(col,1.0);
}
```



```
float castShadow(vec3 pos, vec3 light_pos) {
    float estimated = length(pos - light_pos);
    vec3 dir = normalize(light_pos - pos);
    float actual = traceScene(pos                , dir).x;
    if (actual + 2.f * EPS < estimated)
        return 0.f;
    return 1.f;
}
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Normalized pixel coordinates (from 0 to 1)
    vec2 uv = fragCoord/iResolution.y;
    float aspect = iResolution.x / iResolution.y;

    vec3 from = vec3(0, 0, 0);
    vec3 dir = normalize(vec3(uv - 0.5 * vec2(aspect, 1), 1.0));
    vec4 p = traceScene(from, dir);

    vec3 pos = from + p.x * dir;
    vec3 col = p.yzw;

    vec3 light_pos = vec3(1.f + sin(iTime), 1.f, 1.f);
    //col *= shading(pos, light_pos);
    //col *= castShadow(pos, light_pos);
    col *= min(0.2 + shading(pos, light_pos)
                   * castShadow(pos, light_pos), 1.f);

    // Output to screen
    fragColor = vec4(col,1.0);
}
```

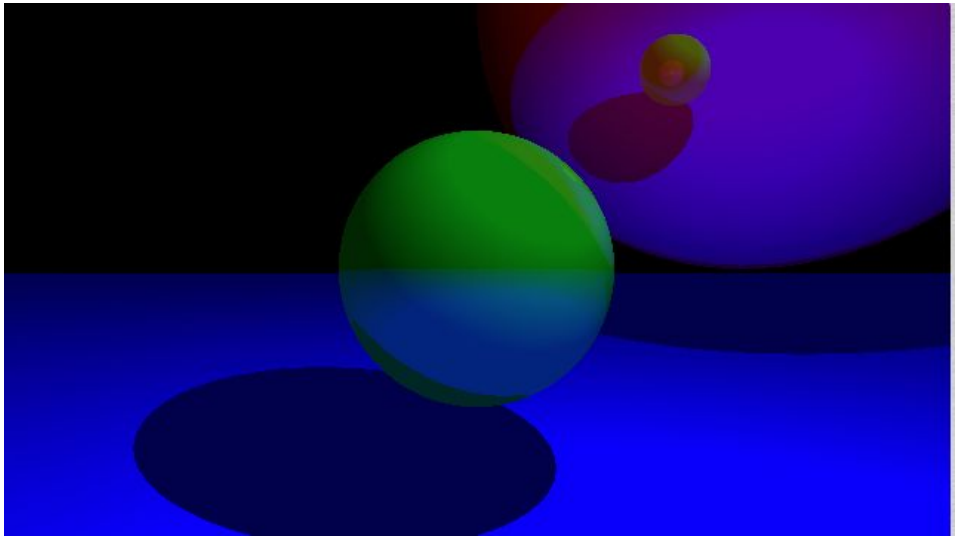


```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Normalized pixel coordinates (from 0 to 1)
    vec2 uv = fragCoord/iResolution.y;
    float aspect = iResolution.x / iResolution.y;

    vec3 from = vec3(0, 0, 0);
    vec3 dir = normalize(vec3(uv - 0.5 * vec2(aspect, 1), 1.0));

    vec3 light_pos = vec3(1.f + sin(iTime), 1.f, 1.f);

    vec3 res_col = vec3(0, 0, 0);
    float weight = 1.0;

    int bounce = 0;
    while (bounce < NBOUNCE) {

        vec2 p = traceScene(from, dir);
        vec3 pos = from + p.x * dir;
        vec3 col = getColor(int(p.y));

        col *= min(0.3 + shading(pos, light_pos)
                       * castShadow(pos, light_pos), 1.f);

        float reflectance = getReflectance(int(p.y));

        bool last = (bounce + 1 >= NBOUNCE || reflectance == 0.0);
        if (last)
            reflectance = 0.0;

        res_col += weight * col * (1.0 - reflectance);
        weight *= reflectance;

        bounce++;
        if (last)
            break;

        vec3 n = normal(pos);
        from = pos + EPS * n;
        dir = reflect(dir, n);
    }

    // Output to screen
    fragColor = vec4(res_col,1.0);
}
```



https://www.shadertoy.com/view/wccyzM

```glsl
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    // Normalized pixel coordinates (from 0 to 1)
    vec2 uv = fragCoord/iResolution.y;
    float aspect = iResolution.x / iResolution.y;

    vec3 from = vec3(0, 0, 0);
    vec3 dir = normalize(vec3(uv - 0.5 * vec2(aspect, 1), 1.0));

    vec3 light_pos = vec3(1.f + sin(iTime), 1.f, 1.f);

    vec3 res_col = vec3(0, 0, 0);
    float weight = 1.0;

    int bounce = 0;
    while (bounce < NBOUNCE) {

        vec2 p = traceScene(from, dir);
        vec3 pos = from + p.x * dir;
        vec3 col = getColor(int(p.y));

        col *= min(0.3 + shading(pos, light_pos)
                       * castShadow(pos, light_pos), 1.f);

        float reflectance = getReflectance(int(p.y));

        bool last = (bounce + 1 >= NBOUNCE || reflectance == 0.0);
        if (last)
            reflectance = 0.0;

        res_col += weight * col * (1.0 - reflectance);
        weight *= reflectance;

        bounce++;
        if (last)
            break;

        vec3 n = normal(pos);
        from = pos + EPS * n;
        dir = reflect(dir, n);
    }

    // Output to screen
    fragColor = vec4(res_col,1.0);
}
```



https://www.shadertoy.com/view/wccyzM

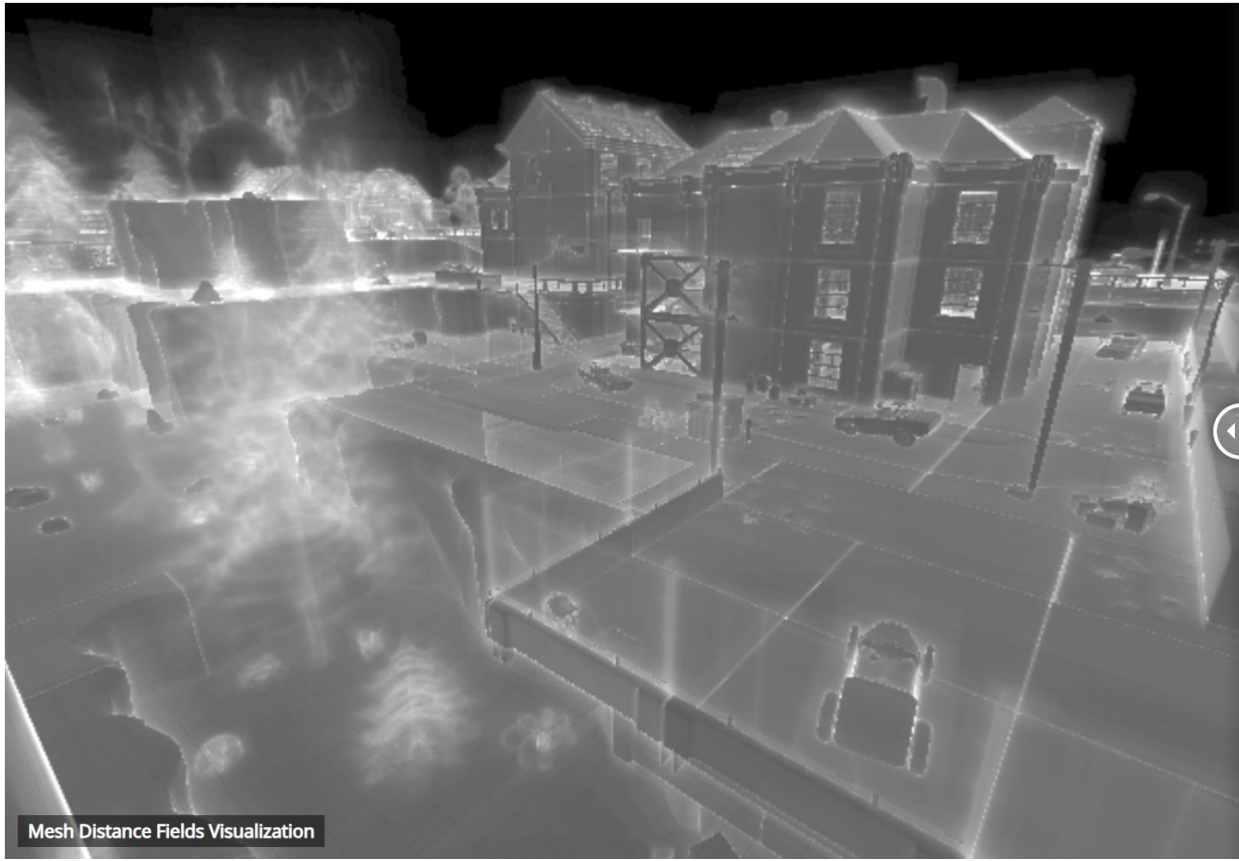
color = $(1-r_0)c_0 + r_0((1-r_1)c_1 + r_1(\ldots)) =$
$= \ldots + r_0(1-r_1)c_1$

r_0 , когда несколько отражений, превращается в $r_0 r_1 r_2 \ldots$ == weight

Lumen (Unreal Engine 5)

В глобальном освещении для ускорения трассировки луча используется Ray Marching по глобальной SDF сцены

Хранение в 3D текстуре с мип-уровнями и стримингом



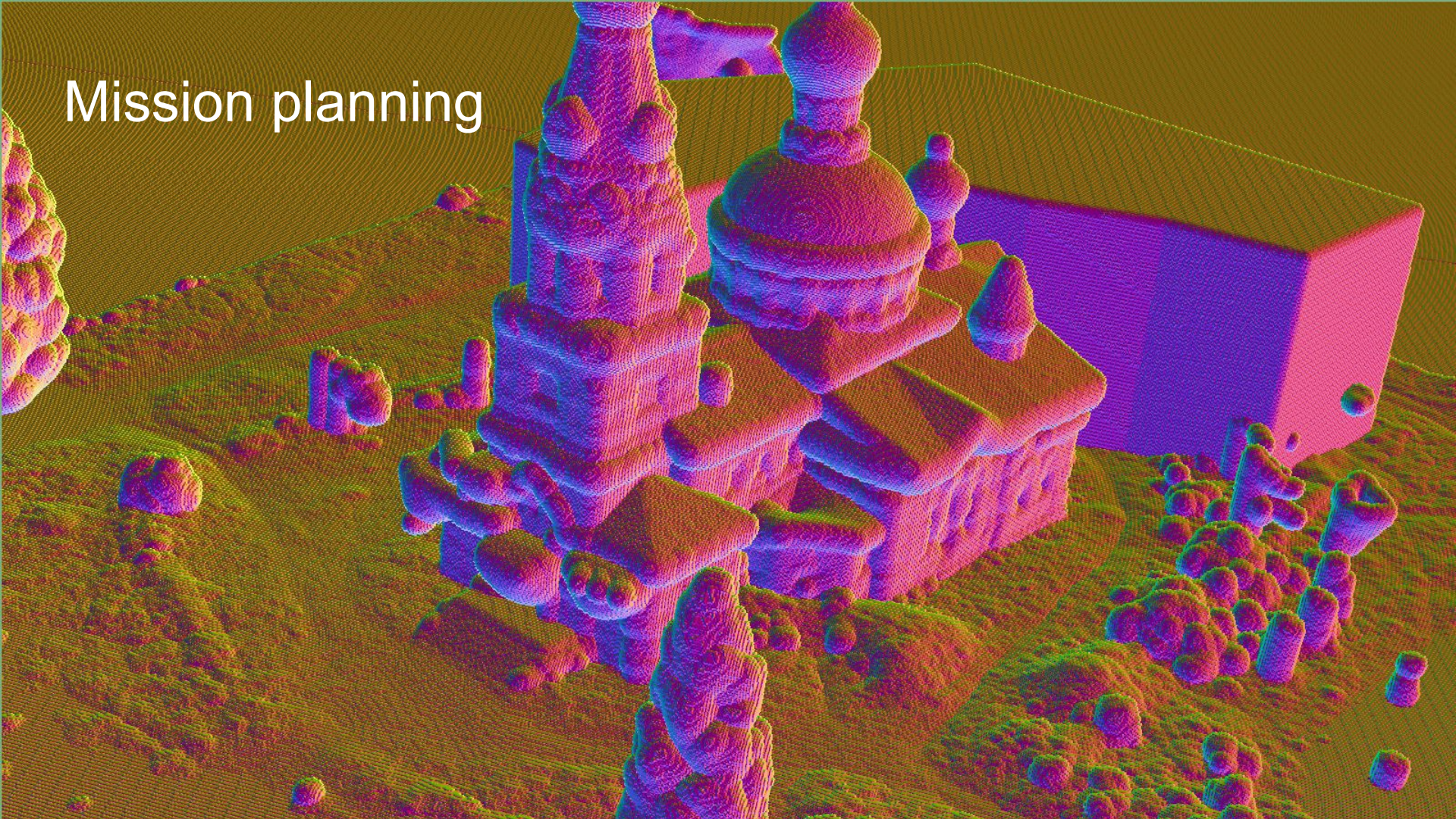
Mesh Distance Fields in Unreal Engine

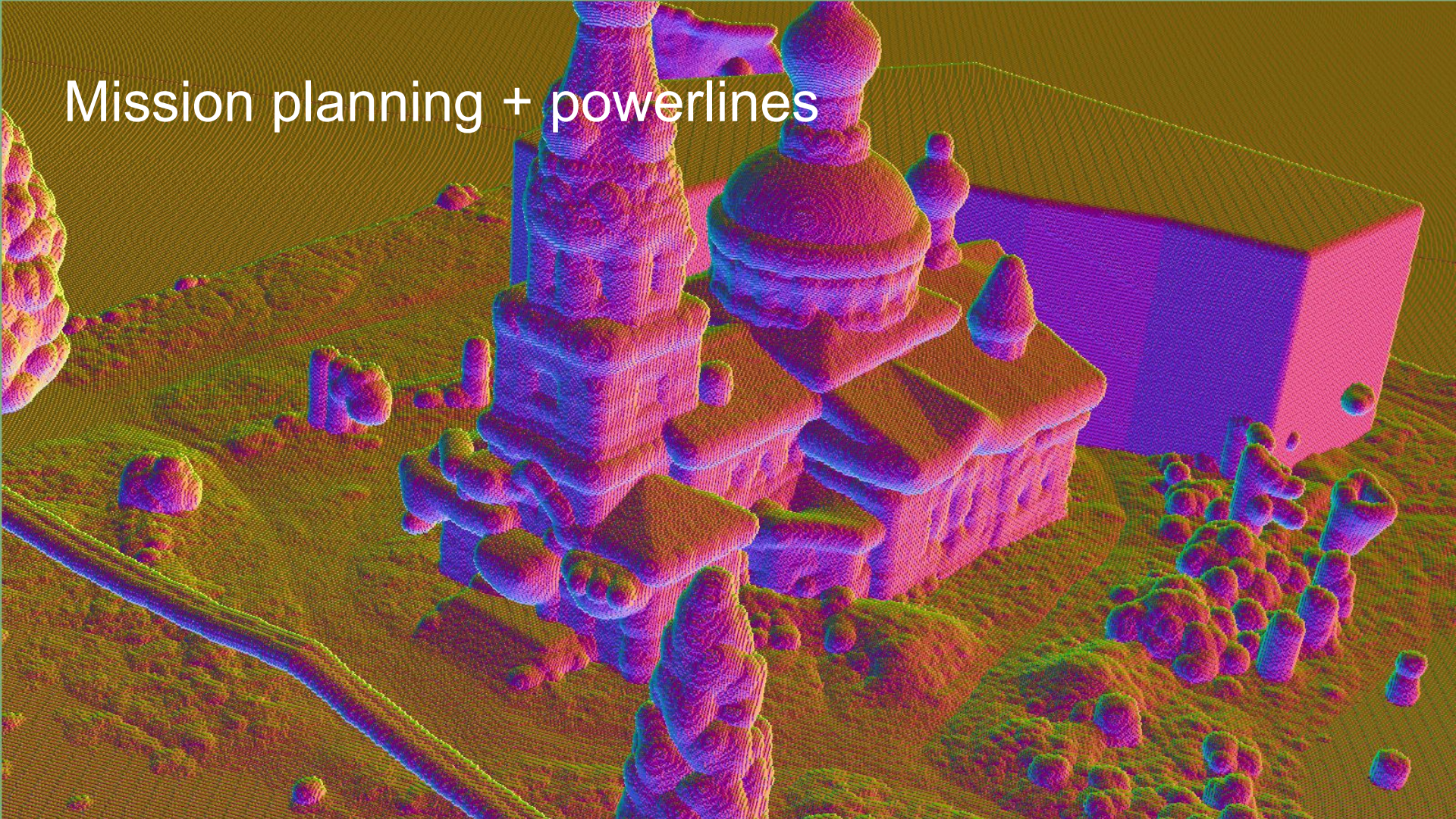
Mission planning

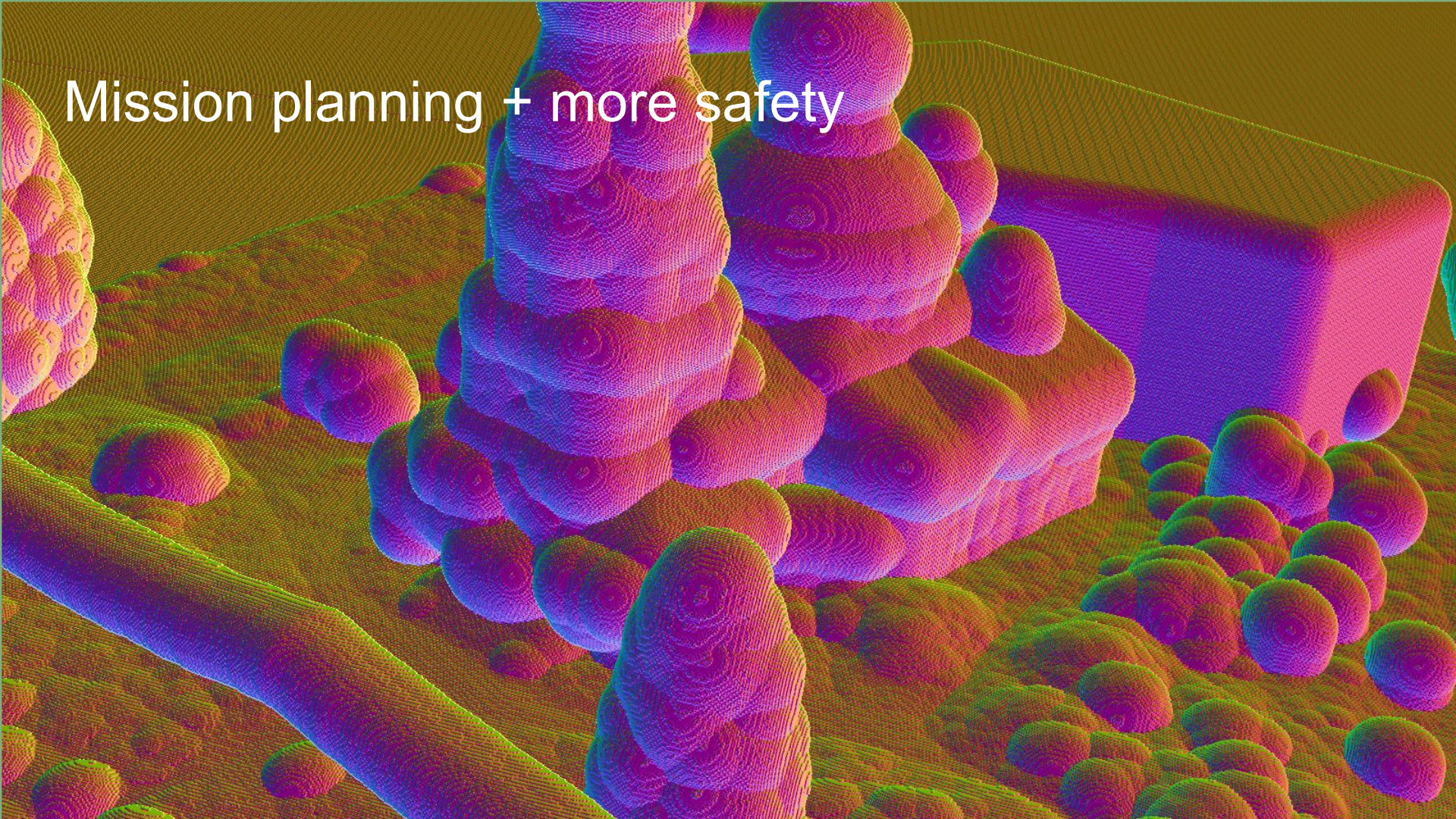
- Задача прокладывания оптимального маршрута дрона по известной местности
- Опасности: здания, деревья, провода
- Зоны запрета полетов, максимальная высота полета (в помещении)

Mission planning

- Задача прокладывания оптимального маршрута дрона по известной местности
- Опасности: здания, деревья, провода
- Зоны запрета полетов, максимальная высота полета (в помещении)
- SDF!
  - Для полигональных моделей зданий и проводов - KD-tree
  - Для разрешенных и запрещенных зон: аналитическая SDF трехмерной призмы
  - Инвертировали знак SDF - разрешили летать только внутри призмы

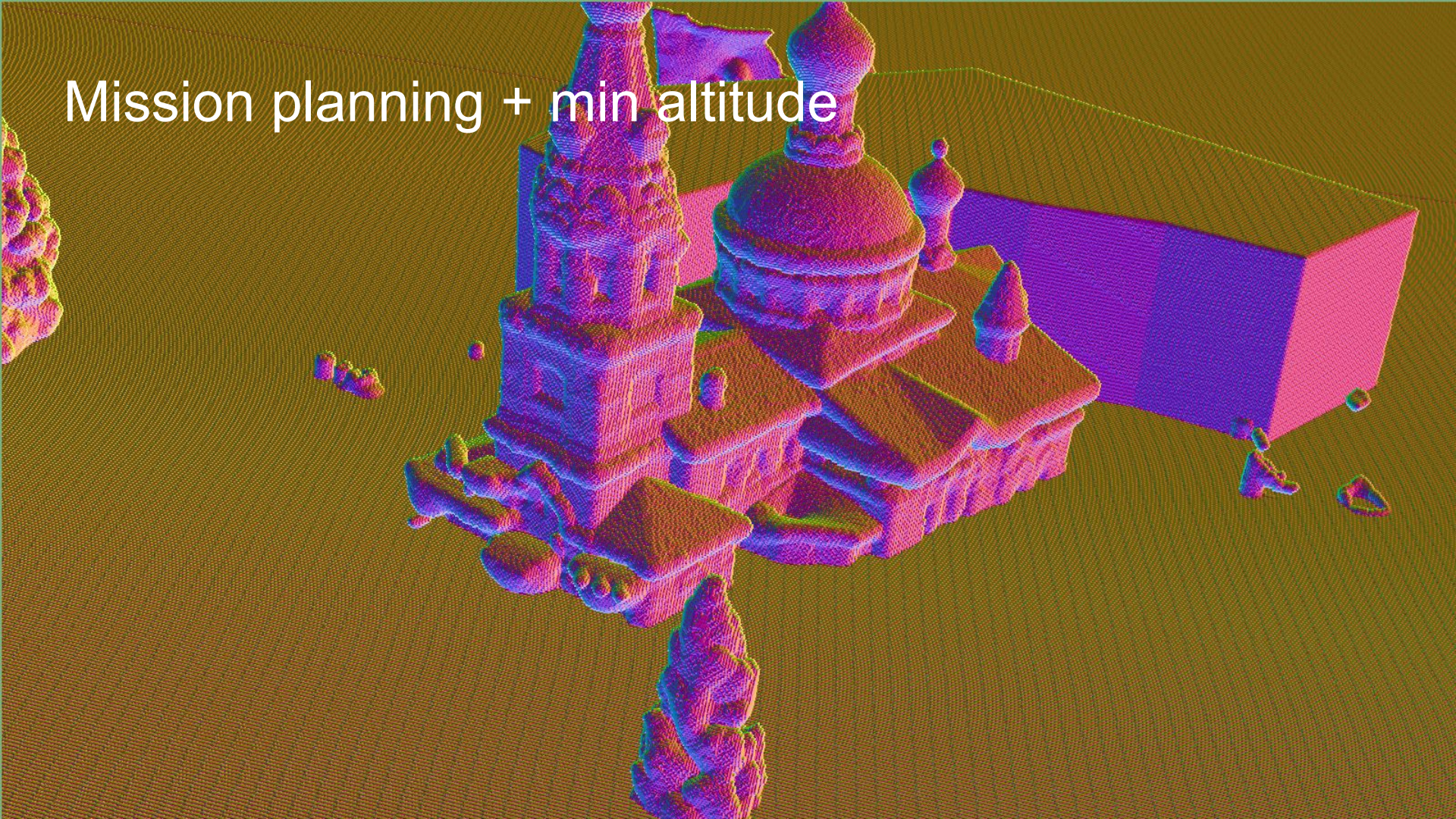

Mission planning


Mission planning + powerlines

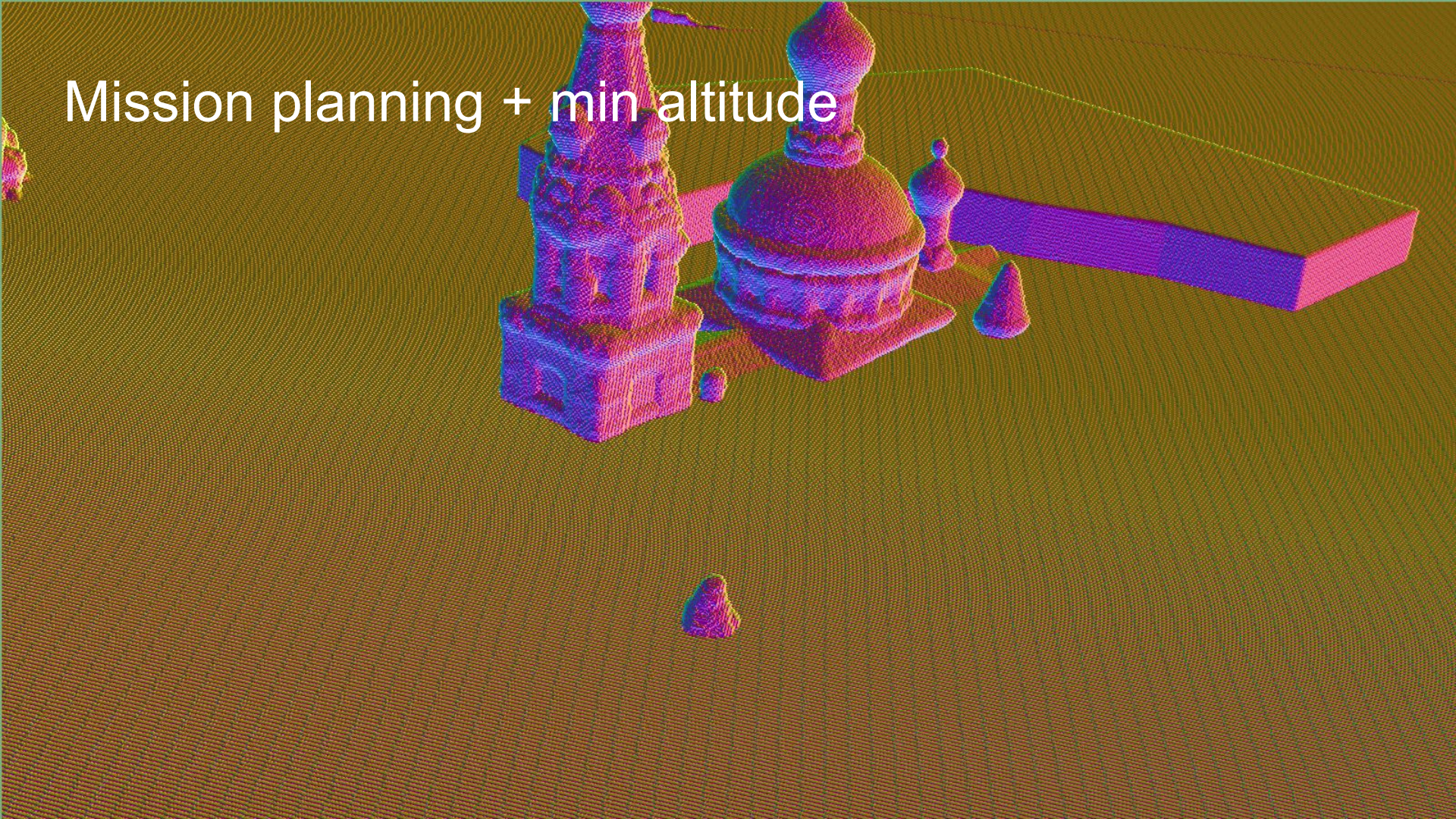

Mission planning + more safety


Mission planning + even more safety

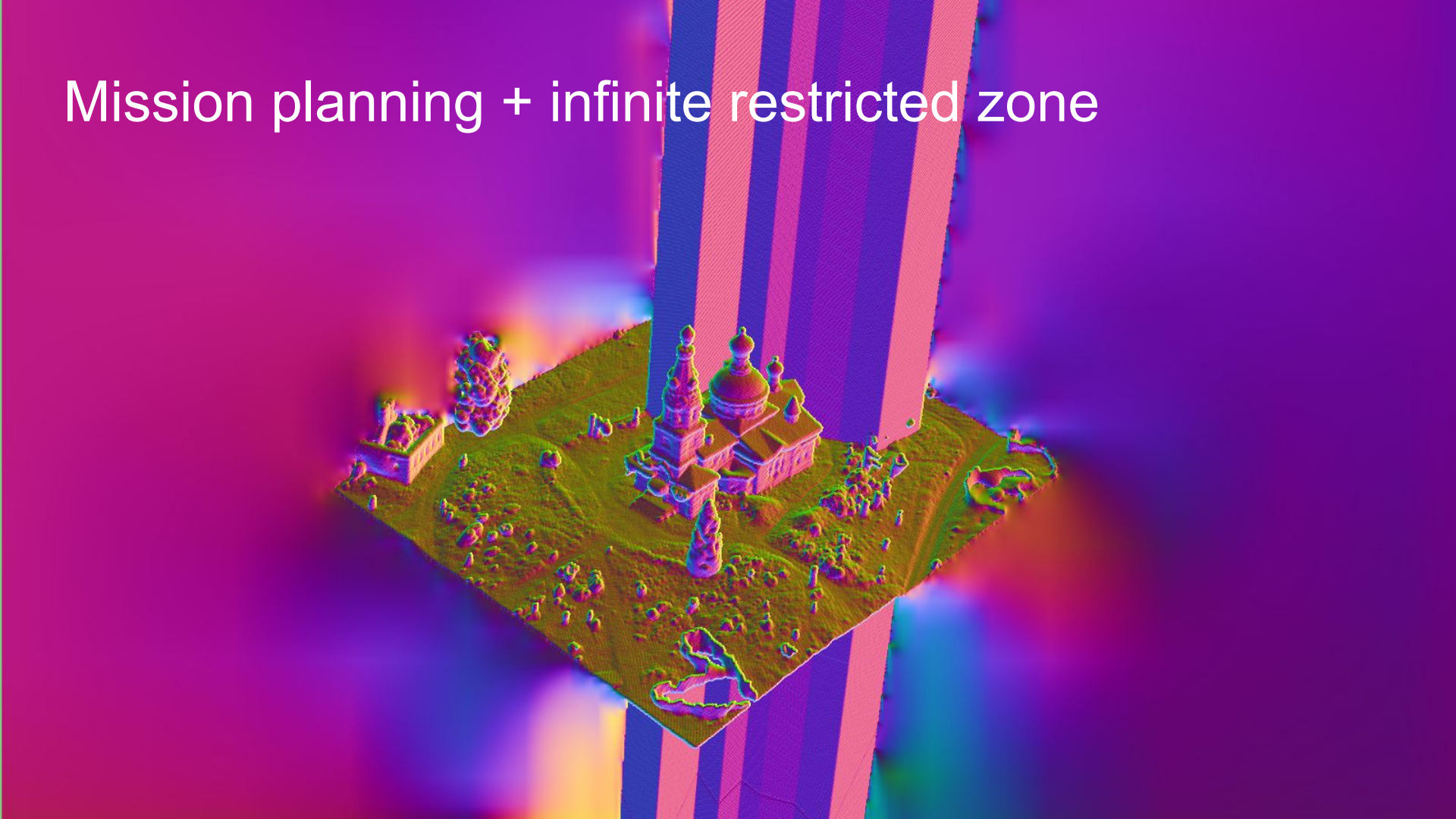
Mission planning + min altitude



Mission planning + min altitude



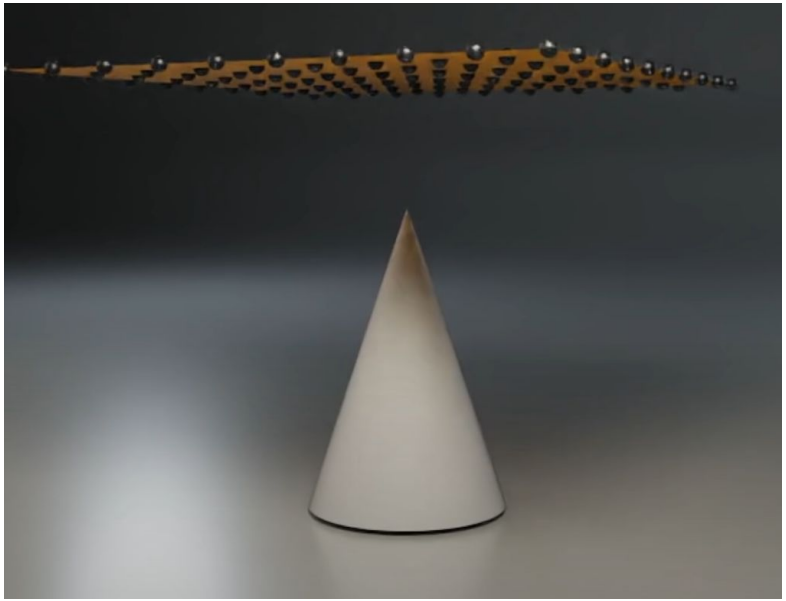
Mission planning + infinite restricted zone



Mission planning + finite allowed altitude

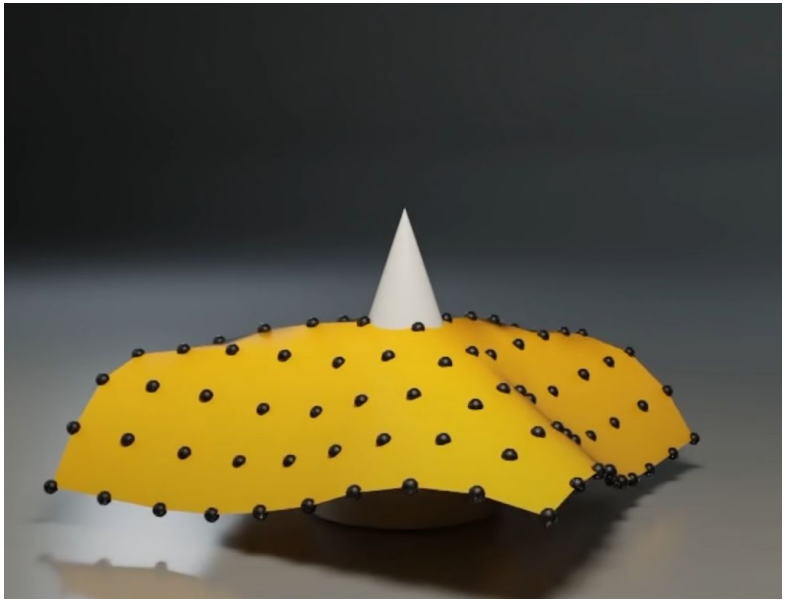
Cloth simulation

- Симуляция тканей
- Нужно находить и обрабатывать коллизии



Cloth simulation

- Симуляция тканей
- Нужно находить и обрабатывать коллизии
- При sampling методе острые объекты могут протыкать ткань



Cloth simulation

- Симуляция тканей
- Нужно находить и обрабатывать коллизии
- При sampling методе острые объекты могут протыкать ткань
- Хочется “векторизовать” ткань, чтобы не приходилось семплировать слишком густо

Cloth simulation

- Пусть ткань состоит из треугольников
- Хотим пересечь SDF с треугольником

Local Optimization for Robust Signed Distance Field Collision

https://youtu.be/ooZ9rUYOFI4

Cloth simulation

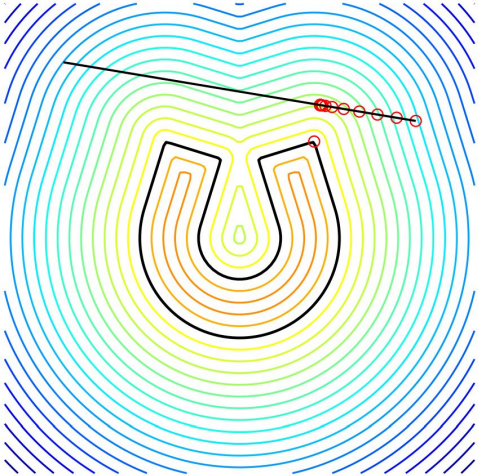
- Пусть ткань состоит из треугольников
- Хотим пересечь SDF с треугольником

- Параметризуем точку внутри барицентрическими координатами
- Нужно найти минимум SDF на треугольнике

$$\begin{aligned} \underset{u,v,w}{\operatorname{argmin}} \quad & \phi(u\mathbf{p} + v\mathbf{q} + w\mathbf{r}) \\ \text{s.t.} \quad & u, v, w \geq 0 \\ & u + v + w = 1. \end{aligned}$$

[Local Optimization for Robust Signed Distance Field Collision]

https://youtu.be/ooZ9rUYOFI4



https://www.shadertoy.com/view/wdcGDB

Cloth simulation

[Local Optimization for Robust Signed Distance Field Collision]

https://youtu.be/ooZ9rUYOFI4

- Пусть ткань состоит из треугольников
- Хотим пересечь SDF с треугольником

- Параметризуем точку внутри барицентрическими координатами
- Нужно найти минимум SDF на треугольнике

- Projected Gradient Descent: делаем шаг в сторону минимизации и возвращаемся на поверхность

$$\begin{aligned} \underset{u,v,w}{\text{argmin}} \quad & \phi(u\mathbf{p} + v\mathbf{q} + w\mathbf{r}) \\ \text{s.t.} \quad & u, v, w \geq 0 \\ & u + v + w = 1. \end{aligned}$$

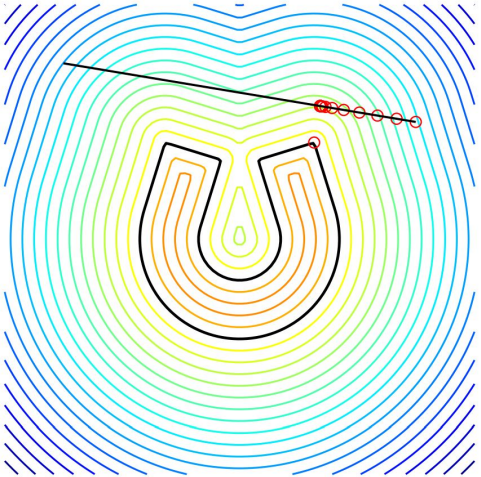
$$\mathbf{d} = \begin{bmatrix} \frac{\partial \phi}{\partial u} & \frac{\partial \phi}{\partial v} & \frac{\partial \phi}{\partial w} \end{bmatrix}^T = \begin{bmatrix} \nabla\phi(\mathbf{x})\mathbf{p} \\ \nabla\phi(\mathbf{x})\mathbf{q} \\ \nabla\phi(\mathbf{x})\mathbf{r} \end{bmatrix}$$

$$\mathbf{c}_{i+1} \leftarrow \mathbb{P}(\mathbf{c}_i - \alpha\mathbf{d}).$$

$$\varphi(\vec{x}) = \varphi(u\vec{p} + v\vec{q} + w\vec{r})$$

$$\frac{\partial \varphi(\vec{x})}{\partial u} = \frac{\partial \varphi(\vec{x})}{\partial \vec{x}} \cdot \frac{\partial \vec{x}}{\partial u} = \nabla\varphi \cdot \vec{p}$$

$$\left[\frac{\partial \varphi(x(u), y(u), z(u))}{\partial u} = \dots \right]$$



https://www.shadertoy.com/view/wdcGDB

Cloth simulation

[Local Optimization for Robust Signed Distance Field Collision]

https://youtu.be/ooZ9rUYOFI4

- Пусть ткань состоит из треугольников
- Хотим пересечь SDF с треугольником

- Параметризуем точку внутри барицентрическими координатами
- Нужно найти минимум SDF на треугольнике

- Projected Gradient Descent: делаем шаг в сторону минимизации и возвращаемся на поверхность
  - Что если попали в локальный минимум?

$$\begin{aligned} \underset{u,v,w}{\text{argmin}} \quad & \phi(u\mathbf{p} + v\mathbf{q} + w\mathbf{r}) \\ \text{s.t.} \quad & u, v, w \geq 0 \\ & u + v + w = 1. \end{aligned}$$

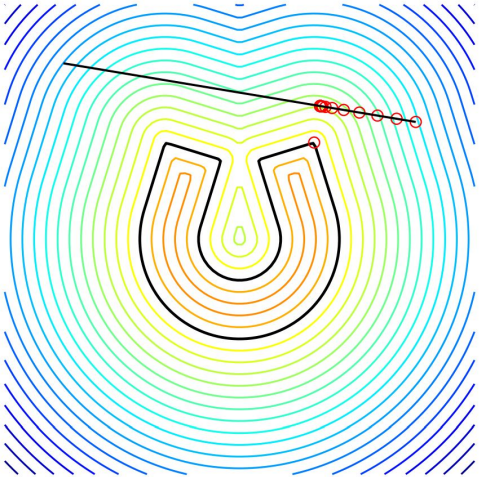
$$\mathbf{d} = \begin{bmatrix} \frac{\partial \phi}{\partial u} & \frac{\partial \phi}{\partial v} & \frac{\partial \phi}{\partial w} \end{bmatrix}^T = \begin{bmatrix} \nabla\phi(\mathbf{x})\mathbf{p} \\ \nabla\phi(\mathbf{x})\mathbf{q} \\ \nabla\phi(\mathbf{x})\mathbf{r} \end{bmatrix}$$

$$\mathbf{c}_{i+1} \leftarrow \mathbb{P}(\mathbf{c}_i - \alpha\mathbf{d}).$$

$$\varphi(\vec{x}) = \varphi(u\vec{p} + v\vec{q} + w\vec{r})$$

$$\frac{\partial \varphi(\vec{x})}{\partial u} = \frac{\partial \varphi(\vec{x})}{\partial \vec{x}} \cdot \frac{\partial \vec{x}}{\partial u} = \nabla\varphi \cdot \vec{p}$$

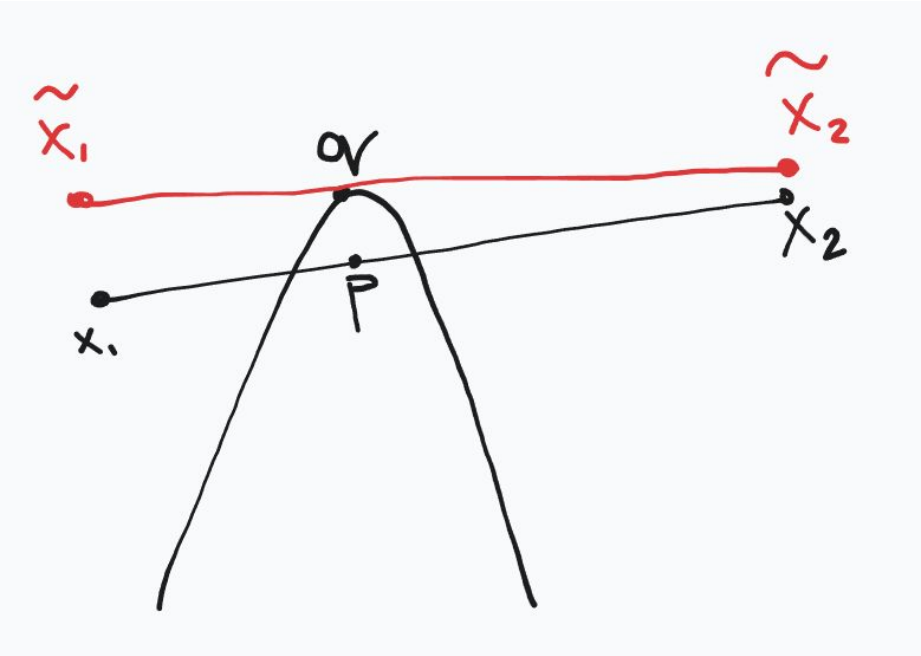
$$\left[\frac{\partial \varphi(x(u), y(u), z(u))}{\partial u} = \dots \right]$$



https://www.shadertoy.com/view/wdcGDB

Cloth simulation

Нашли “самую глубокую” точку **p** на сегменте (треугольнике).
Как обновить координаты?



Cloth simulation

Нашли “самую глубокую” точку **p** на сегменте (треугольнике).
Как обновить координаты?

Как рассчитать **q**? (у нас препятствие задано sdf)

$$p = c_1 x_1 + c_2 x_2$$

$$\Delta = q - p$$

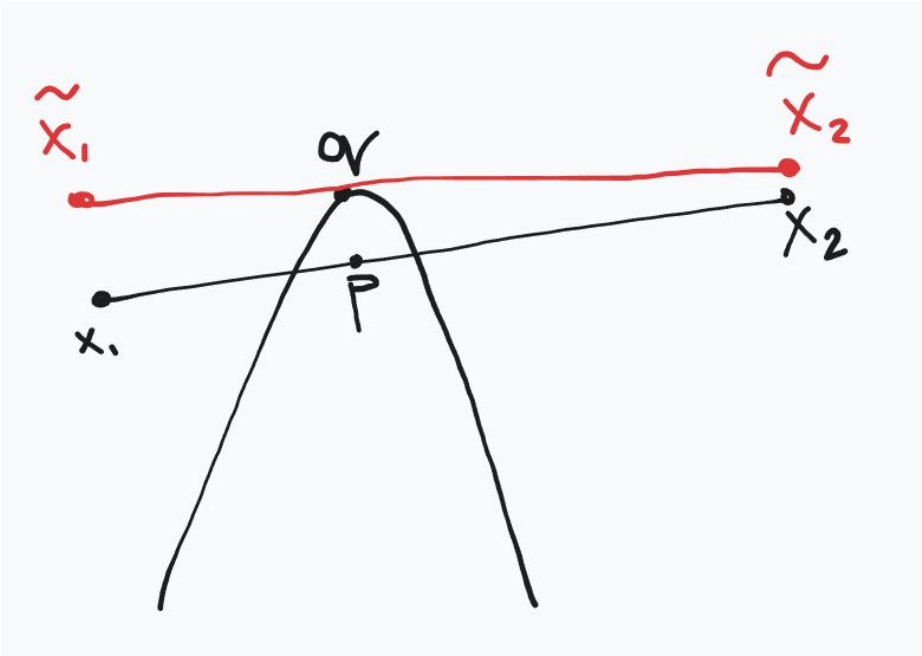
$$d = c_1^2 + c_2^2$$

s:

$$\tilde{x}_1 \leftarrow x_1 + \frac{c_1}{d}\Delta$$

$$\tilde{x}_2 \leftarrow x_2 + \frac{c_2}{d}\Delta$$

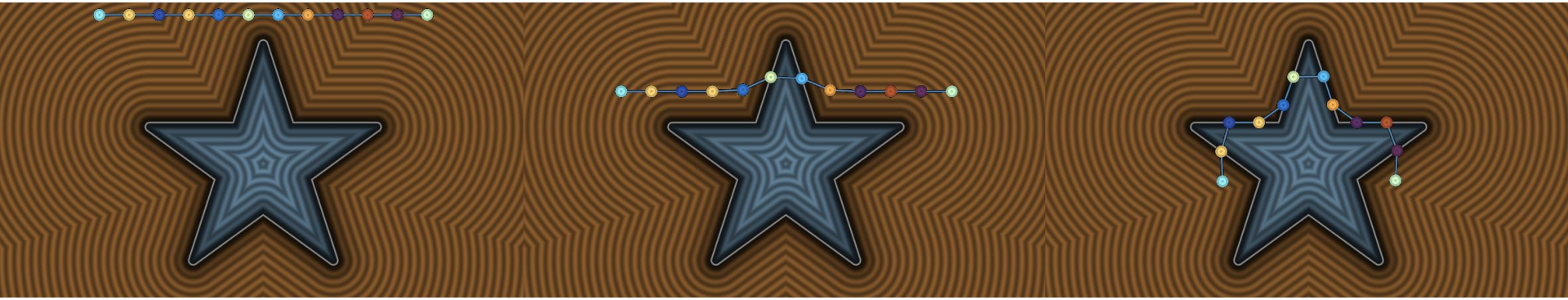
$$c_1\tilde{x}_1 + c_2\tilde{x}_2 = q$$



Jakobsen T., Advanced Character Physics

Without constraints

https://www.shadertoy.com/view/tfcczN

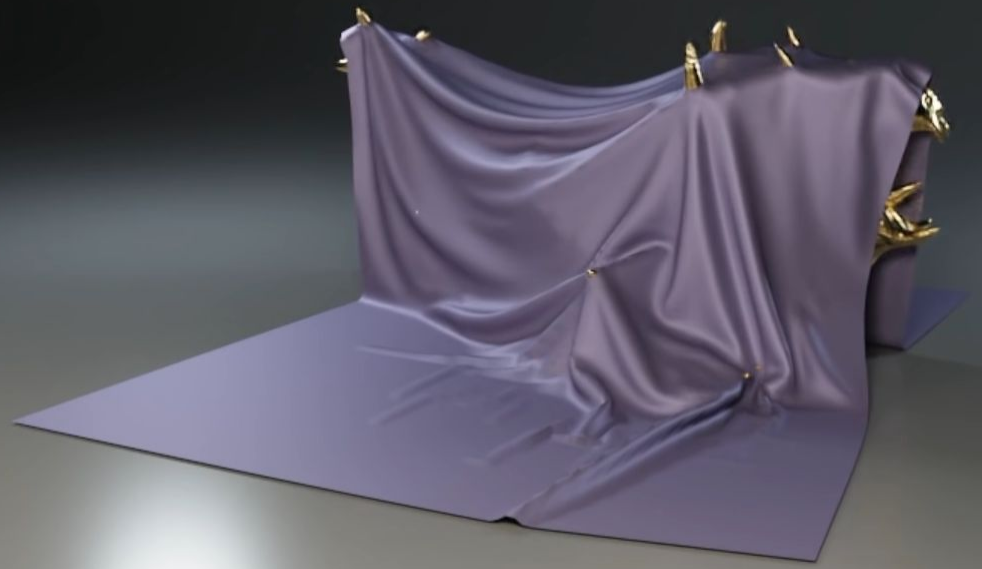


With constraints

Previous technique

Cloth simulation

https://youtu.be/ooZ9rUYOFI4



[Macklin et al. 2020]

Source

New technique

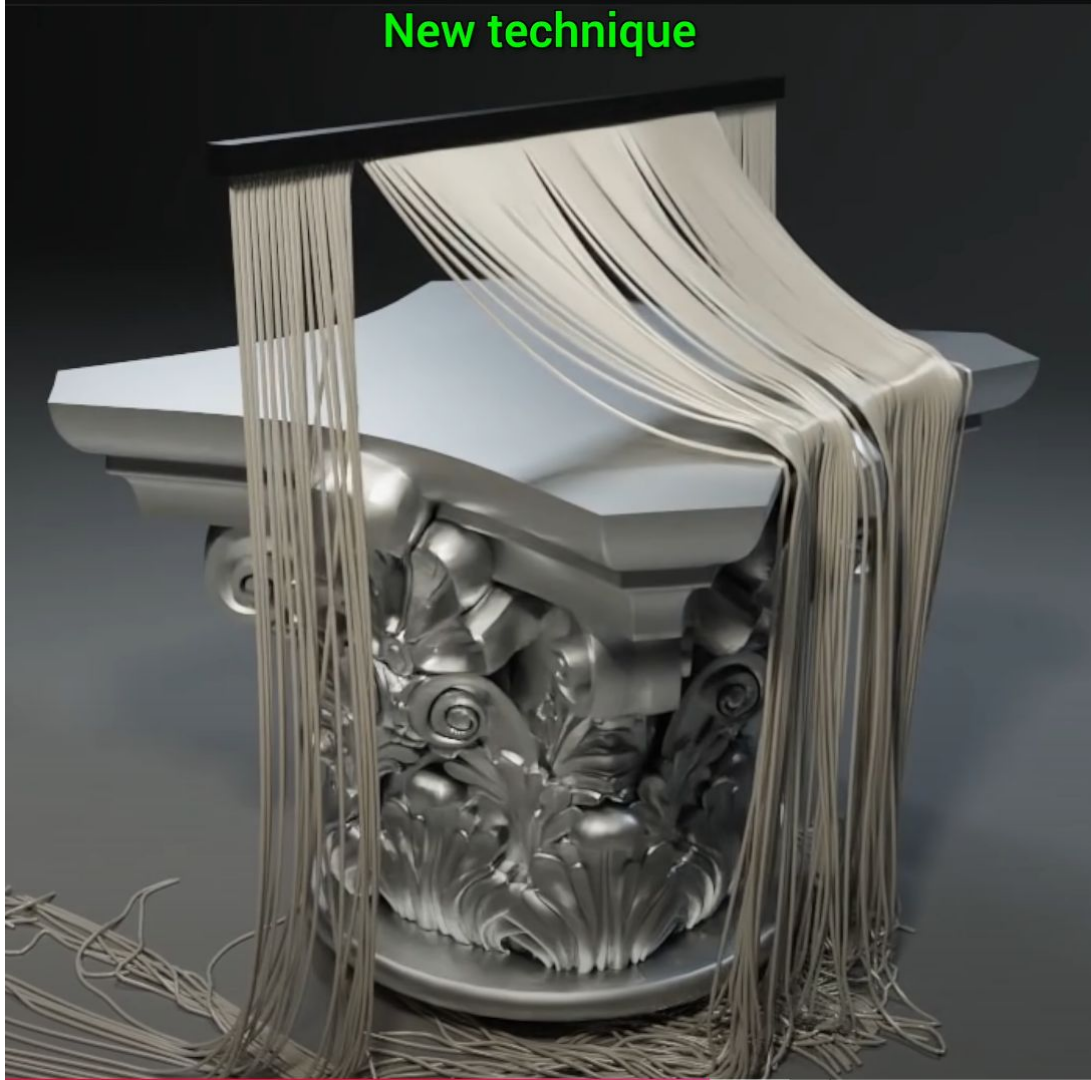
Cloth simulation

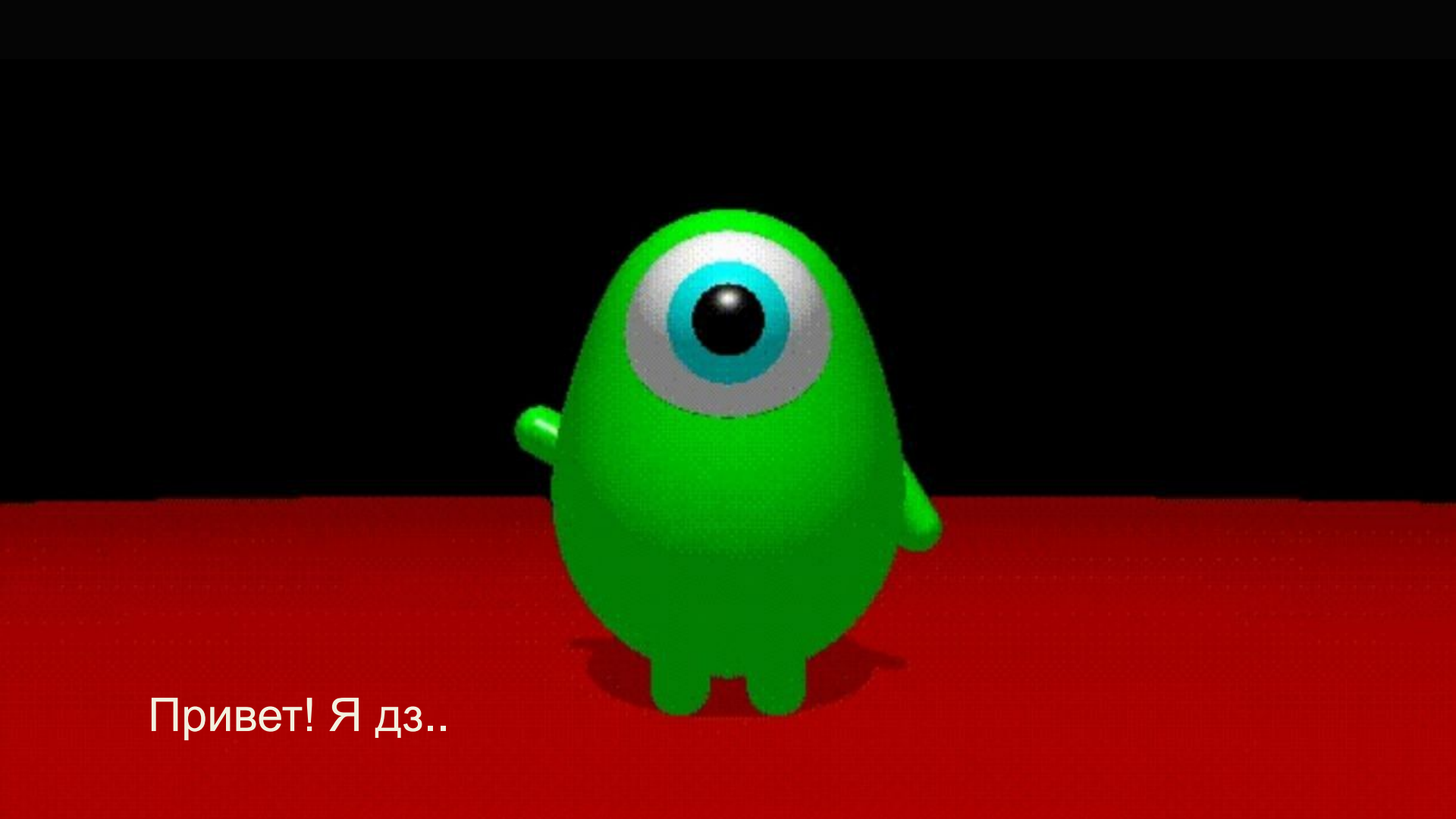
[Macklin et al. 2020]

Source

Cloth simulation

Cloth simulation




Привет! Я дз..