

[Pages](#) / ... / [Newcomer internal training](#)

2. Working with normals

Created by Anton Hritsan, last modified on Sep 15, 2021

Table of content

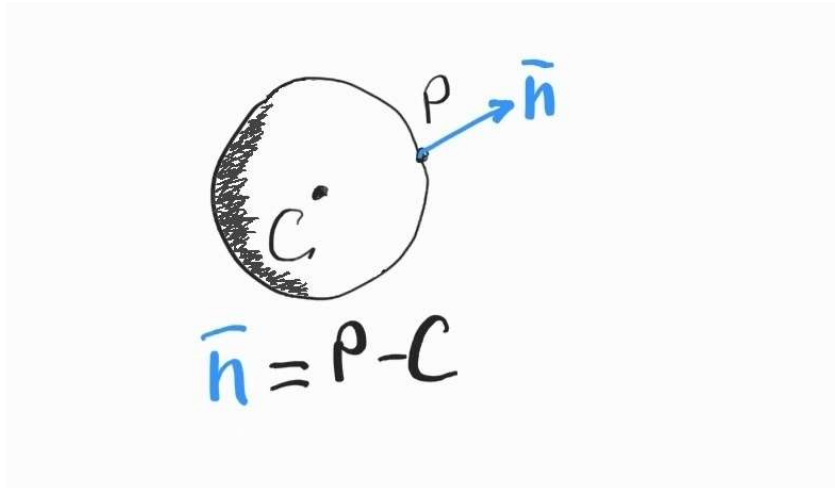
- [Problem and solution](#)
- [Normals](#)
- [Visualization](#)
- [Tasks](#)
 - [You need to visualize normals from 'far' part of sphere.](#)
 - [Treat sphere as Box, and show Box's normals in sphere](#)

Problem and solution

Your current code work almost perfect except of few facts. You don't know what part of sphere ray intersects with. Also, flat color doesn't look like natural, due to lack of lighting and shading. Before you start to implement all there merry things, we need to deal with normals. This part of training will cover normals and an idea to visualize them.

Normals

Normal - it's a vector that perpendicular to surface at any point on surface. Sphere - it's an ideal model for normal calculation, because of sphere's nature. Geometrically, sphere is represented as object with unit vector as radius. It gives us that fact, every normal in sphere can be calculated as difference between point in surface with center of object. Illustration below



Please pay extra attention normal is director, and taking normal vector normalized - very good idea.

Visualization

Well, normals is directional, and direction is a vector. Every vector can be treated as color. Your goal - draw normal as result color!

But before you will do it, you need to know a position in sphere to calculate normal. As far as we want to see the nearest part of surface, we need to all finding such point in our hit_sphere code.

Naïve implementation will be like

```
float3 Application::RayTracing( const Ray& r )
{
    // hardcoded place of sphere
    float3 fSpherePos = { 0.0f, 0.0f, 1.0f };

    auto hit_sphere = []( const float3 & center,
    {
        float3 AC = r.getPosition() - center;
        float a = float3::Dot( r.getDirection(), AC );
        float b = 2.0f * float3::Dot( AC, r.getDirection() );
        float c = float3::Dot( AC, AC ) - r.getRadius() * r.getRadius();
        float discriminant = b * b - 4.0f * a * c;

        if( discriminant < 0.0f )
        {
            return -1.0f;
        }

        float sol1 = ( -b + sqrtf( discriminant ) ) / ( 2.0f * a );
        float sol2 = ( -b - sqrtf( discriminant ) ) / ( 2.0f * a );

        float sol = Math::Min( sol1, sol2 );

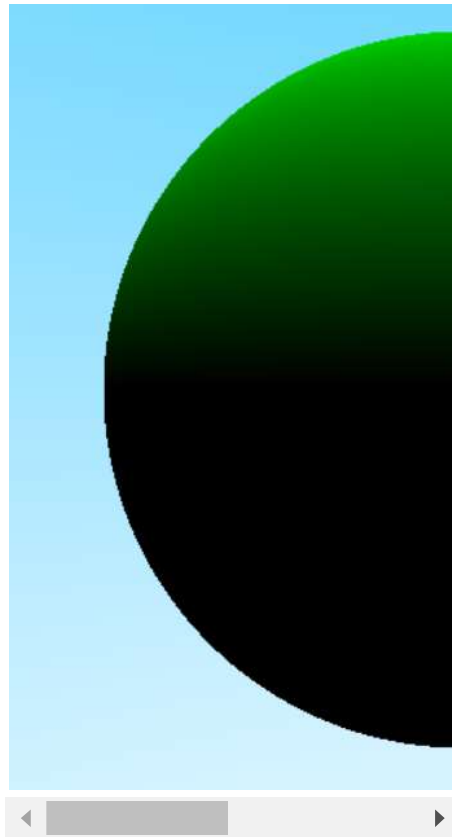
        return sol;
    };

    float fDistance = hit_sphere( fSpherePos, r );

    if( fDistance > 0.0f )
    {
        float3 posAt = ray.PointAt( fDistance );
        float3 normal = ( posAt - fSpherePos );

        return normal;
    }

    // nothin' hit, horizon
    //
    return this->getHorizonColor( ray );
}
}
```



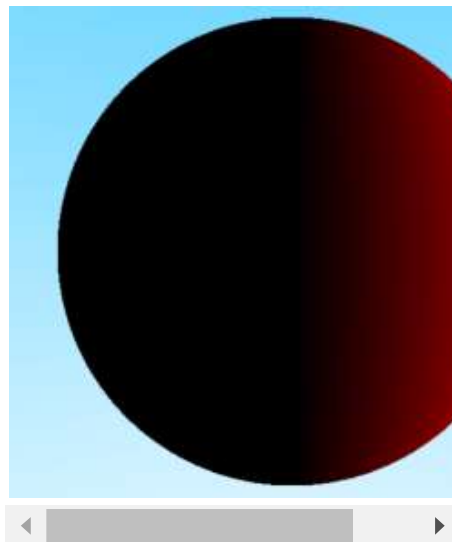
Well, it doesn't seem correct. a lot of black color here. Let's try to debug it. I will disable parts of color from normal to visual each channel separately

Red channel only: (values by X axis)

```
if( fDistance > 0.0f )
{
    float3 posAt = ray.PointAt( fDistance );
    float3 normal = ( posAt - fSpherePos );

    // blocking here
    normal.y = 0.0f;
    normal.z = 0.0f;

    return normal;
}
```



Green channel only: (values by Y axis)

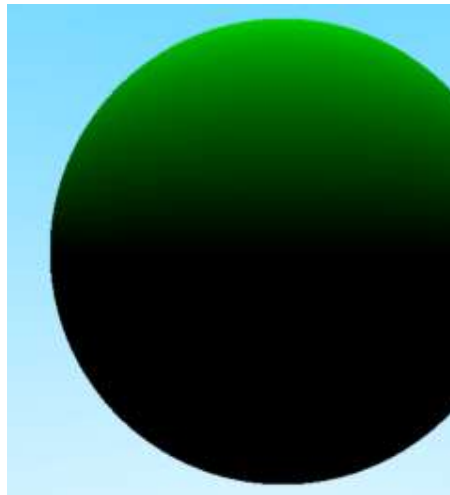
```

if( fDistance > 0.0f )
{
    float3 posAt = ray.Po
    float3 normal = ( pos

    // blocking here
    normal.x = 0.0f;
    normal.z = 0.0f;

    return normal;
}

```



And Blue channel only: (Values by Z axis)

```

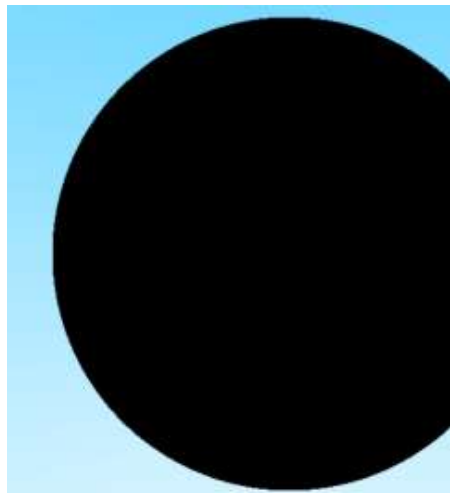
float fDistance = hit_spher

if( fDistance > 0.0f )
{
    float3 posAt = ray.Poin
    float3 normal = ( posAt

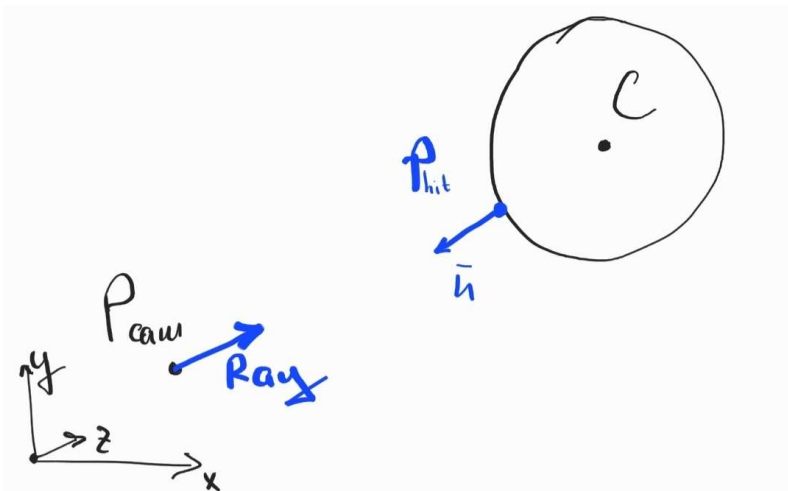
    // blocking here
    normal.x = 0.0f;
    normal.y = 0.0f;

    return normal;
}

```



Oh well, we have 'wrong' Z direction! Actually no, we have right Z direction for our Lefthand coordinate system. According to description of lefthand system, Z direction points to forward, so if we draw geometrically our current situation we would get next:



our normal is looking to us, just with negative 'z' value. Every normalized normals are in range of $[-1.. 1]$. For us it means, to debug 'z' direction we need to invert it. Let's try it

```

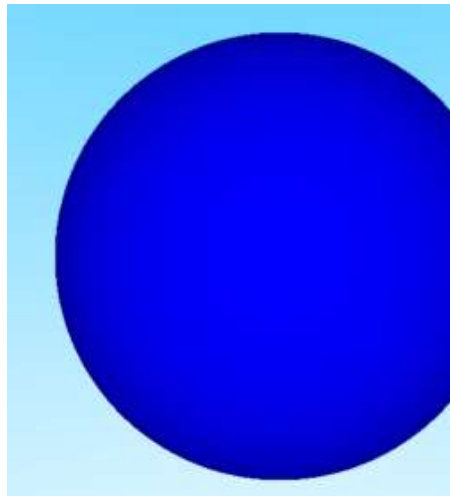
if( fDistance > 0.0f )
{
    float3 posAt = ray.PointAt( f
    float3 normal = ( posAt - f

    // ONLY FOR VISUAL!!!!
    normal.z = -normal.z;

    // blocking here
    normal.x = 0.0f;
    normal.y = 0.0f;

    return normal;
}

```

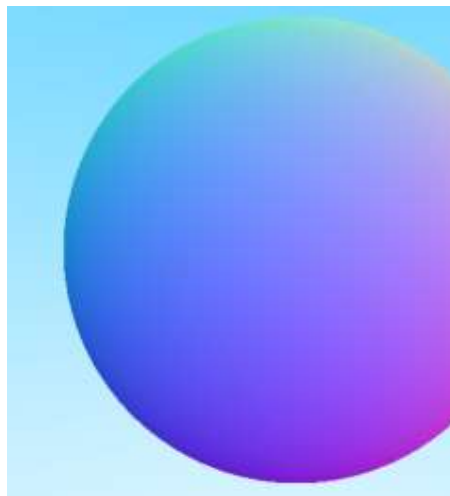


Negative values will not be shown in colors, so, converting normal to range [0.. 1] we would produce next

```

1 if( fDistance > 0.0f )
2 {
3     float3 posAt = ray.PointAt( f
4     float3 normal = ( posAt - fSh
5
6     // ONLY FOR VISUAL!!!!
7     normal.z = -normal.z;
8
9     // blocking here
10    //normal.x = 0.0f;
11    //normal.y = 0.0f;
12
13    // normal in randge [-1,1]
14    // converting it to [0,1]
15    return ( 0.5f * ( normal + fl
16
17 }

```



Tasks

You need to visualize normals from 'far' part of sphere.

Treat sphere as Box, and show Box's normals in sphere

As far as you can't show every normal as-is due to negative values, prepare a list of unique color for each box's side. Such list can be like

```

// cube side colors
float3 color[ 6 ] =
{
    float3( 1,0,0 ),    // right
    float3( 0,0,1 ),    // front
    float3( 1,0,1 ),    // left
    float3( 0,1,1 ),    // back

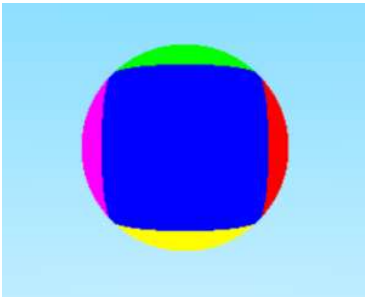
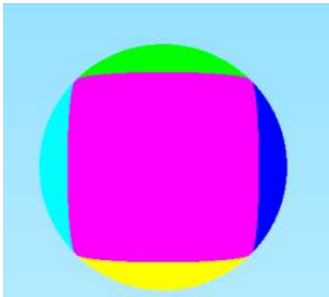
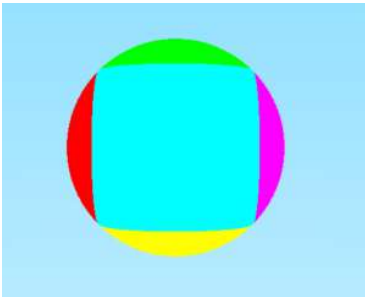
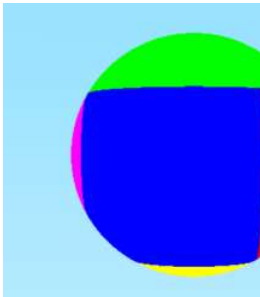
```

```
float3( 0,1,0 ),    // top
float3( 1,1,0 )     // bottom
};
```

Also, be aware of sphere's normal. it's inverted by 'z' direction(in our left hand coordinate system)! After some primitive trigonometry rules, you can determine side of box by normal.
I understand, this task sounds like it doesn't have any sense, but with such functional you can make interesting behavior with zero model in applications, just math. For example: skybox\skydome, visualization sphere in tern of bounding(like AABB or OBB).

Hint: I Highly recommend you to draw in paper your trigonometry before coding! With proper image this task IS trivial!

Sample result

to front(zero angles)	horizontal angle 90 degree	horizontal angle 180 degree	mixed. hor. angle -15 +15
			

No labels

