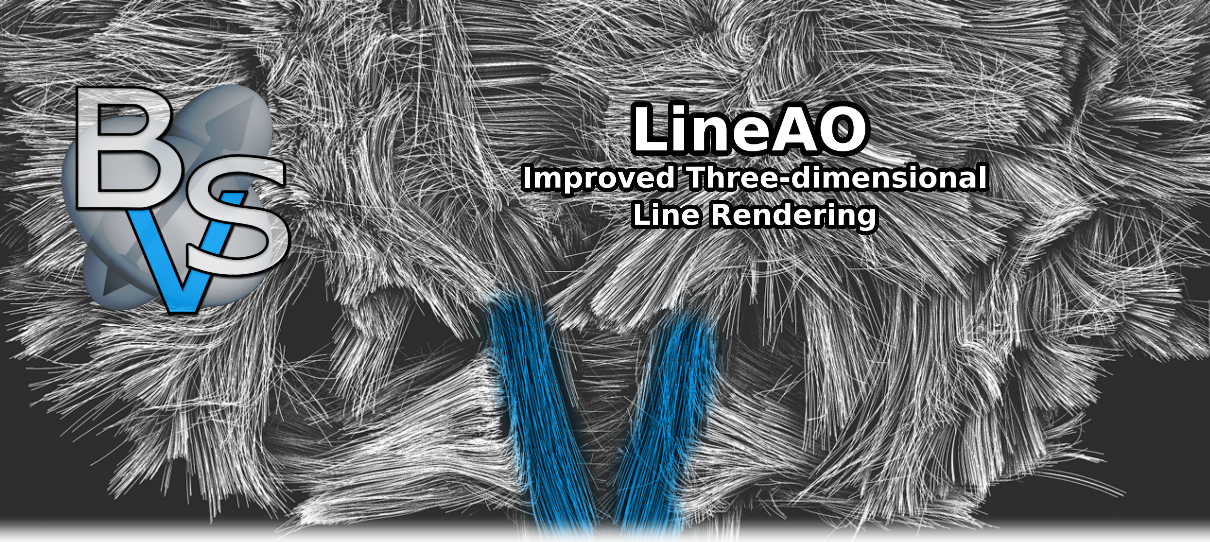


LineAO
Improved Three-dimensional Line Rendering

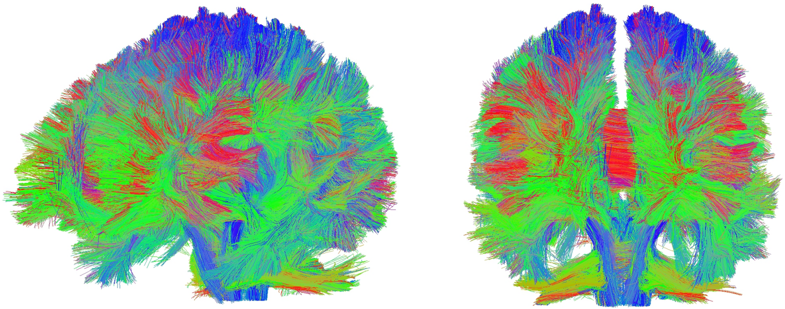
Sebastian Eichelbaum[1] Mario Hlawitschka[2] Gerik Scheuermann[1]

[1] Image and Signal Processing Group, University of Leipzig, Germany
[2] Scientific Visualization Group, University of Leipzig, Germany

Why? — State-of-the-Art

- Isn't standard line rendering sufficient for line data exploration?

Why? — Plain Coloring



(a) Side (b) Front

Figure : Tractography data of a human brain: 5m single lines — Do you see relations between bundles of lines? Do you see lobes and fissures?

Why? — Plain Coloring - The Problem

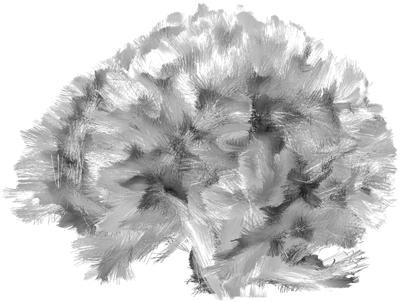
- Colors can provide coarse directional information:
  - IFF you are used to the coloring and know its meaning
    - What to do if the color encodes some other feature in the data?
  - IFF you are used to this certain type of dataset
    - What to do if not?

  → Because you have a mental image of this data
- Spatial relations and shape can only be seen by interacting with the scene!

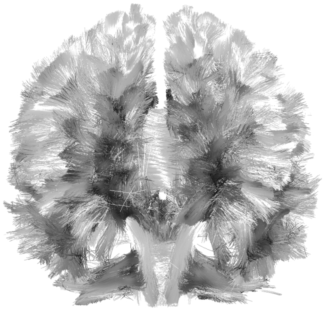
Why? — Plain Coloring - The Problem

- Possible solution: shape from shading.
    - See Ramachandran et al.
    - Shading in computer graphics?
        - local illumination provides structure
        - global illumination provides relative, spatial information
        - → Let's try!

V. S. Ramachandran. Perception of shape from shading. *Nature*, 331:163–166, 1988.



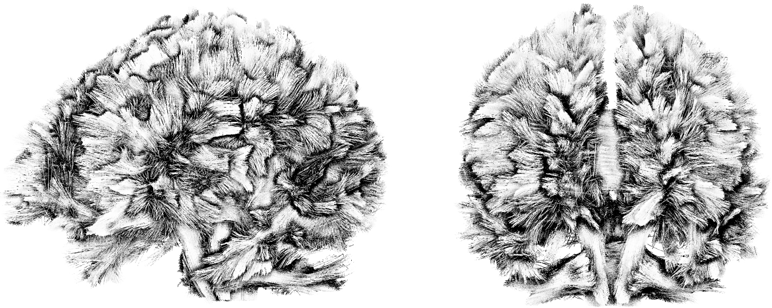
(a) Side



(b) Front

Figure : The illuminated lines approach (Zöckler et al. 1996, Mallo et al. 2005) can help to grasp global structures due to specular highlights, but provides no spatial relations.

Why? — Screen Space Ambient Occlusion



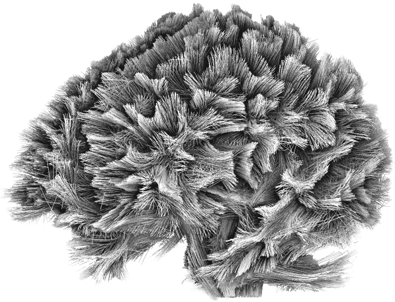
(a) Side (b) Front

Figure : The ambient occlusion approach from CryEngine 2 (Mittring 2007) provides some spatial information, but is not able to handle very thin objects accurately.

Why? — Limitations

- Spatial relations only via interaction
- Current SSAO approaches do not work properly with thin geometry

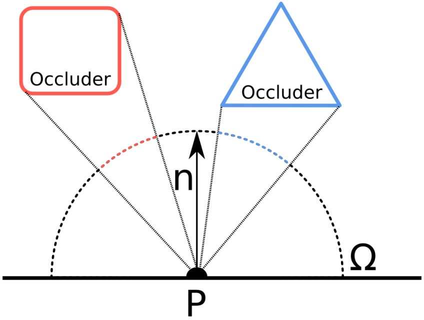
$\Rightarrow$ LineAO provides a solution!



(a) Side

(b) Front

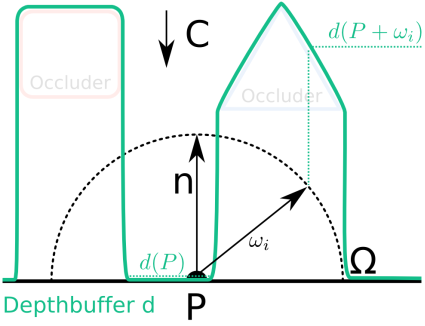
Figure : LineAO provides *global and local structure* as well as *spatial relations in bundles and between bundles* without the need for interaction.



- Defined for each point P on each surface of the scene
- Surface normal n at P defines hemisphere Ω
- AO is the amount of hemisphere surface occluded by other objects

- $AO(P, n) = \frac{1}{\pi} \int_{\Omega} (1 - V(\omega, P)) \langle \omega, n \rangle d\omega,$
- Calculation of visibility function V costly

How? — Screen Space Ambient Occlusion

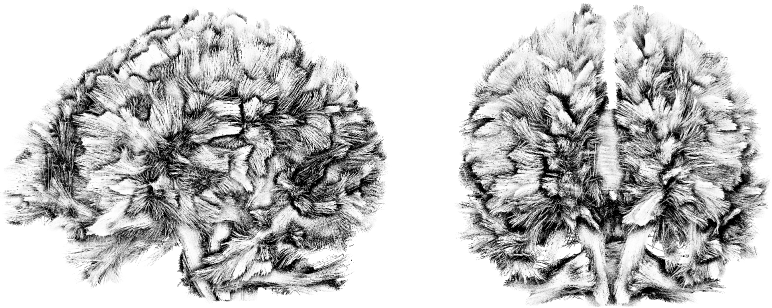


- Discretized problem to solve in screen space
- Randomly sample the hemisphere S-times at multiple ω_i
- Utilize depth difference for visibility check

$$\rightarrow \quad V(\omega, P) = \begin{cases} 1 & \text{if } d(P) - d(P+\omega) < 0 \\ 0 & \text{else,} \end{cases}$$

$$\rightarrow \quad AO_s(P, n) = \frac{1}{s} \sum_{i=1}^{s} (1 - V(\omega_i, P)) \langle \omega_i, n \rangle$$

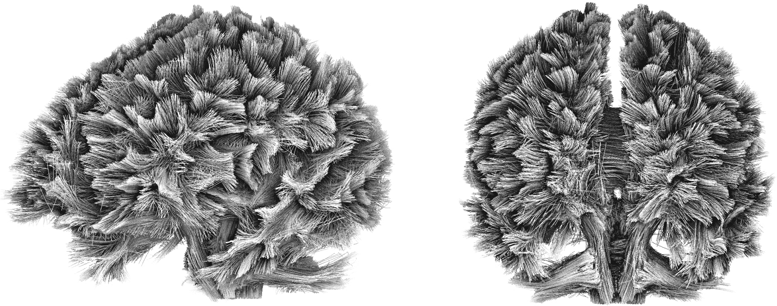
Why? — Screen Space Ambient Occlusion



(a) Side (b) Front

Figure : The ambient occlusion approach from CryEngine 2 (Mittring 2007) provides some spatial information, but is not able to handle very thin objects accurately.

What? — LineAO Introduced



(a) Side (b) Front

Figure : LineAO provides *global and local structure* as well as *spatial relations in bundles and between bundles* without the need for interaction.

How? — LineAO Described

$$LineAO_{s_r,s_h,r_0}(P) = \sum_{j=0}^{s_r-1} AO_{\frac{s_h}{j+1},j}(P, r_0 + jz(P))$$

$$AO_{s,l}(P,r) = \frac{1}{s}\sum_{i=1}^{s}\left[(1 - V_l(r\omega_i, P))g_l(r\omega_i, P)\right]$$

$$V_l(\omega, P) = \begin{cases} 1 & \text{if } d_l(P) - d_l(P+\omega) < 0 \\ 0 & \text{else,} \end{cases}$$

$$g_l(\omega, P) = g_l^{depth}(\omega, P) \cdot g_l^{light}(\omega, P)$$

$$\Delta d_l(\omega, P) = d_l(P) - d_l(P+\omega) \in [-1, 1]$$

$$\delta(l) = \left(1 - \frac{l}{s_r}\right)^2 \in (0, 1]$$

$$h(x) = 3x^2 - 2x^3, \forall x \in [0,1] : h(x) \in [0,1]$$

$$g_l^{depth}(\omega, P) = \begin{cases} 0, & \text{if } \Delta d_l(\omega, P) > \delta(l) \\ 1, & \text{if } \Delta d_l(\omega, P) < \delta_0 \\ 1 - h\left(\frac{d_l(\omega,P) - \delta_0}{\delta(l) - \delta_0}\right), & \text{else.} \end{cases}$$

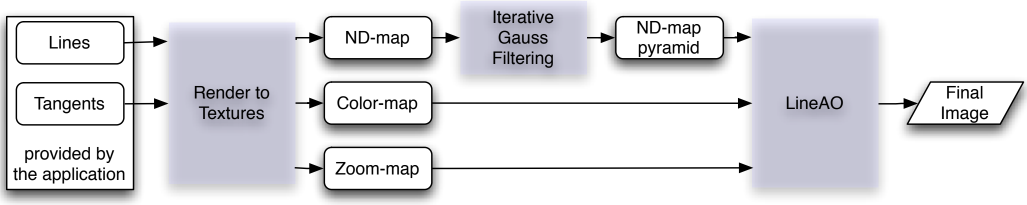
$$L_l(\omega, P) = \sum_{s \in \text{Lights}} BRDF(L_s, l_s, n_l(P), \omega)$$

$$g_l^{\text{light}}(\omega, P) = 1 - min(L_l(\omega, P), 1)$$

```glsl
#define SCALERS WGE_POSTPROCESSOR_LINEAO_SCALERS
#define SAMPLES WGE_POSTPROCESSOR_LINEAO_SAMPLES
const float invSamples = 1.0 / float( SAMPLES );
const float falloff = 0.00001;
vec3 randNormal = normalize( ( texture2D( u_noiseSampler, where * u_noiseSizeX ).xyz * 2.0 ) - vec3( 1.0 ) );
vec3 currentPixelSample = getNormal( where ).xyz;
float currentPixelDepth = getDepth( where );
vec3 ep = vec3( where.xy, currentPixelDepth );
vec3 normal = currentPixelSample.xyz;
float radius = ( getZoom() * u_lineaoRadiusSS / float( u_texture0SizeX ) ) / ( 1.0 - currentPixelDepth );
vec3 ray;
vec3 hemispherePoint;
vec3 occluderNormal;
float occluderDepth;
float depthDifference;
float normalDifference;
float occlusion = 0.0;
float radiusScaler = 0.0;

for( int l = 0; l < SCALERS; ++l )
{
    float occlusionStep = 0.0;
    #define radScaleMin 1.5
    radiusScaler += radScaleMin + l;
    int numSamplesAdded = 0;
    for( int i = 0; i < SAMPLES; ++i )
    {
        vec3 randSphereNormal = ( texture2D( u_noiseSampler, vec2( float( i ) / float( SAMPLES ),
                                                                   float( l + 1 ) / float( SCALERS ) ) ).rgb * 2.0 ) - vec3( 1.0 );
        vec3 hemisphereVector = reflect( randSphereNormal, randNormal );
        ray = radiusScaler * radius * hemisphereVector;
        ray = sign( dot( ray, normal ) ) * ray;
        hemispherePoint = ray + ep;
        if( ( hemispherePoint.x < 0.0 ) || ( hemispherePoint.x > 1.0 ) ||
            ( hemispherePoint.y < 0.0 ) || ( hemispherePoint.y > 1.0 )
          )
        {
            continue;
        }
        numSamplesAdded++;
        occluderDepth = getDepth( hemispherePoint.xy );
        occluderNormal = getNormal( hemispherePoint.xy ).xyz;
        depthDifference = currentPixelDepth - occluderDepth;
        float pointDiffuse = max( dot( hemisphereVector, normal ), 0.0 );
        vec3 t = getTangent( hemispherePoint.xy ).xyz;
        vec3 newnorm = normalize( cross( normalize( cross( t, normalize( hemisphereVector ) ) ), t ) );
        float occluderDiffuse = max( dot( newnorm, gl_LightSource[0].position.xyz ), 0.0 );
        vec3 H = normalize( gl_LightSource[0].position.xyz + normalize( hemisphereVector ) );
        float occluderSpecular = pow( max( dot( H, occluderNormal ), 0.0 ), 100.0 );
        normalDifference =  pointDiffuse * ( occluderSpecular + occluderDiffuse );
        normalDifference = 1.5 - normalDifference;
        float scaler = 1.0 - ( l / ( float( SCALERS - 1 ) ) );
        float densityInfluence = scaler * scaler * u_lineaoDensityWeight;
        float densityWeight = 1.0 - smoothstep( falloff, densityInfluence, depthDifference );
        occlusionStep += normalDifference * densityWeight * step( falloff, depthDifference );
    }
    occlusion += ( 1.0 / float( numSamplesAdded ) ) * occlusionStep;
}
float occlusionScalerFactor = 1.0 / ( SCALERS );
occlusionScalerFactor *= u_lineaoTotalStrength;
return clamp( ( 1.0 - ( occlusionScalerFactor * occlusion ) ), 0, 1 );
```

How? — LineAO Described



- Prepare: lines and tangent data
- LineAO: for each pixel do:
  - Sample surrounding using multiple hemispheres
  - Classify occluders whether they are local or distant occluders
  - Weight according to distance, surface properties and illumination
  - Sum up all weighted occluders

Results — Features

- Greatly improved structural and spatial perception for the rendered line data in a very intuitive and natural way
- Simultaneous depiction of local and global line structures
- Renders in real time without pre-computation
- Consistency under modification and interaction

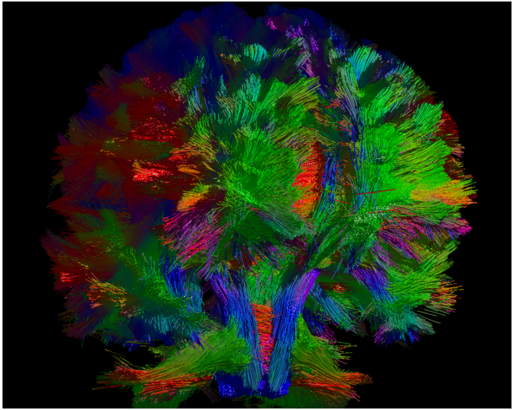
Results — Limitations

- LineAO is not suited for coarse line data
- LineAO does not work for two dimensional and quasi two dimensional data

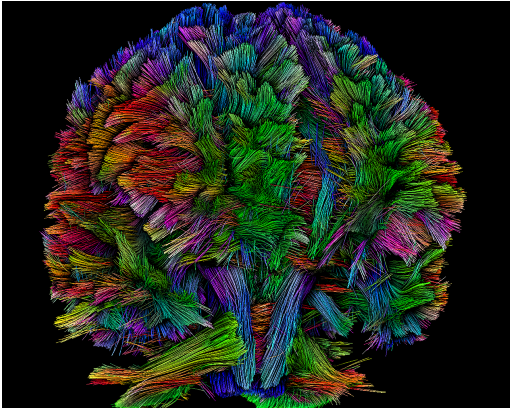
Results — Performance

- LineAO does not depend on dataset complexity
- LineAO works in constant time when compared to dataset complexity
  - LineAO only depends on the size of the screen
- Bottleneck is the GPU’s line geometry processing power

Results — Radiosity versus LineAO

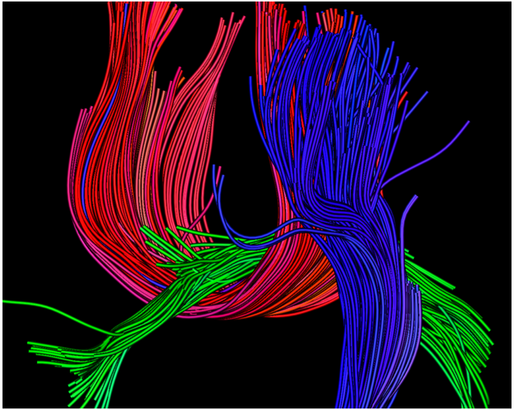


(a) Radiosity - 0.0000185FPS (15h per Frame)

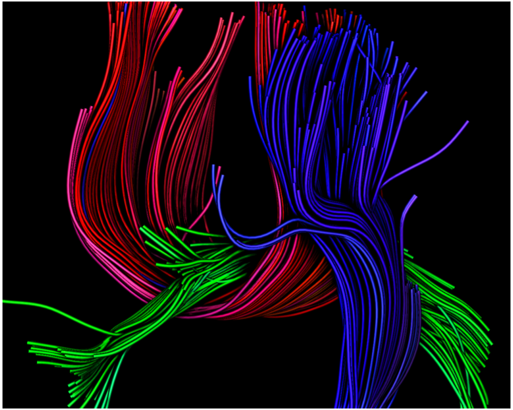


(b) LineAO - 17FPS

Results — Combined with Tube Rendering

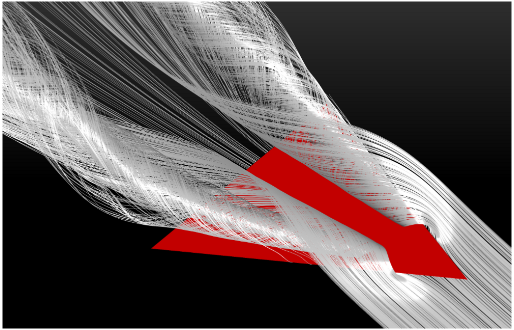


(c) Tube Rendering

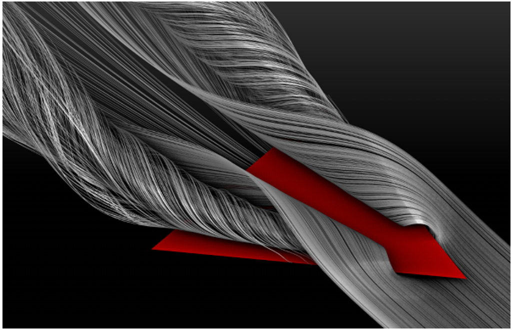


(d) Tube Rendering with LineAO

Results — Combined with Illuminated Streamlines

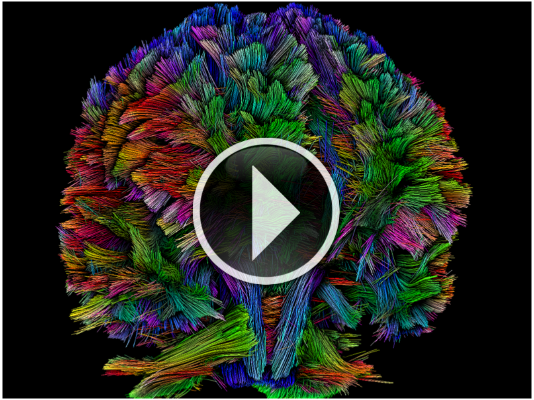


(e) Illuminates Lines



(f) Combined with LineAO

Results — Video



Results — Future Work

- Combination of LineAO with illustrative approaches.
- Adaptive sampling depending on line density in a pixel’s surrounding, while estimating the density in screen-space.

OpenWalnut

Visualization in a Nutshell

Thank You!

Questions?

Details — Weighting

$$LineAO_{s_r,s_h,r_0}(P) = \sum_{j=0}^{s_r-1} AO_{\frac{s_h}{j+1},j}(P, r_0 + jz(P))$$

$$AO_{s,l}(P,r) = \frac{1}{s}\sum_{i=1}^{s}[(1 - V_l(r\omega_i, P))g_l(r\omega_i, P)]$$

$$V_l(\omega, P) = \begin{cases} 1 & \text{if } d_l(P) - d_l(P+\omega) < 0 \\ 0 & \text{else,} \end{cases}$$

$$g_l(\omega, P) = g_l^{depth}(\omega, P) \cdot g_l^{light}(\omega, P)$$

$$\Delta d_l(\omega, P) = d_l(P) - d_l(P+\omega) \in [-1, 1]$$

$$\delta(l) = \left(1 - \frac{l}{s_r}\right)^2 \in (0, 1]$$

$$h(x) = 3x^2 - 2x^3, \forall x \in [0,1] : h(x) \in [0,1]$$

$$g_l^{depth}(\omega, P) = \begin{cases} 0, & \text{if } \Delta d_l(\omega, P) > \delta(l) \\ 1, & \text{if } \Delta d_l(\omega, P) < \delta_0 \\ 1 - h\left(\frac{d_l(\omega,P)-\delta_0}{\delta(l)-\delta_0}\right), & \text{else.} \end{cases}$$

$$L_l(\omega, P) = \sum_{s \in \text{Lights}} BRDF(L_s, l_s, n_l(P), \omega)$$

$$g_l^{\text{light}}(\omega, P) = 1 - min(L_l(\omega, P), 1)$$

- Weight each occluder with $g_l(r\omega_i, P)$
- Classify and weight according to distance and used hemisphere
- Incorporate local light reflected towards occluder, opposing the occlusion due to the added "light-energy"