CS 300: Advanced Computer Graphics I

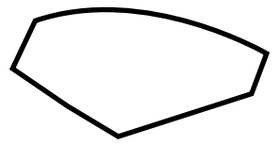
Pushpak Karnick

Lecture 23

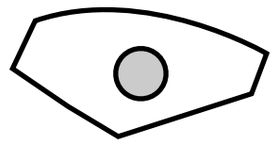
Global Illumination & Ambient Occlusion

Introduction to Global Illumination

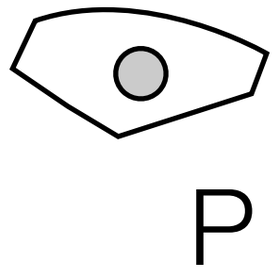
Illumination in Computer Graphics



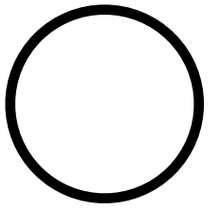
Illumination in Computer Graphics

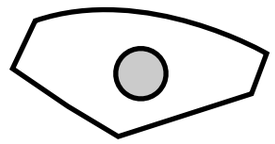


Illumination in Computer Graphics



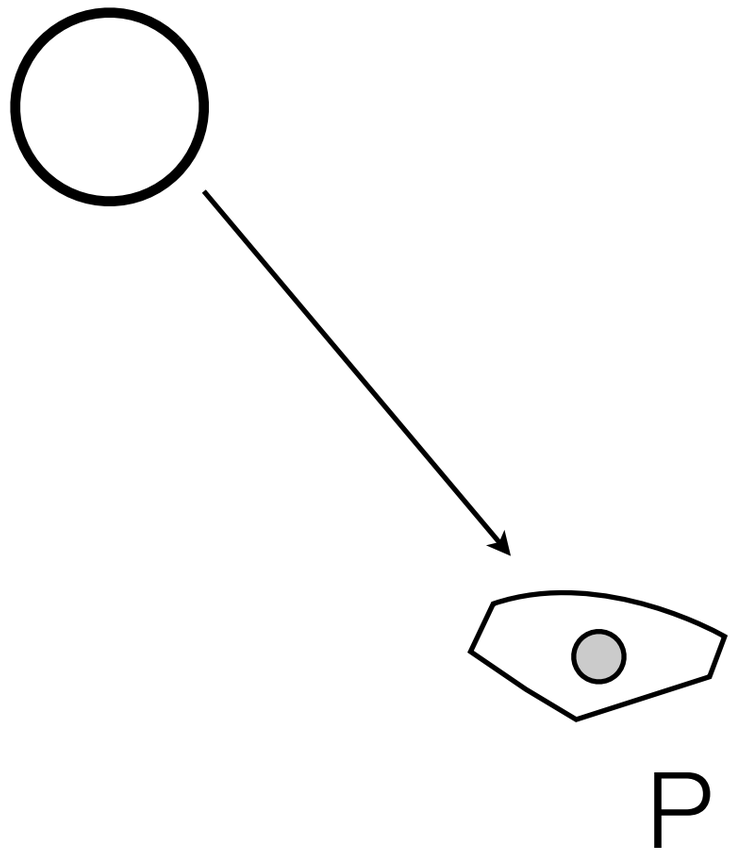
Illumination in Computer Graphics



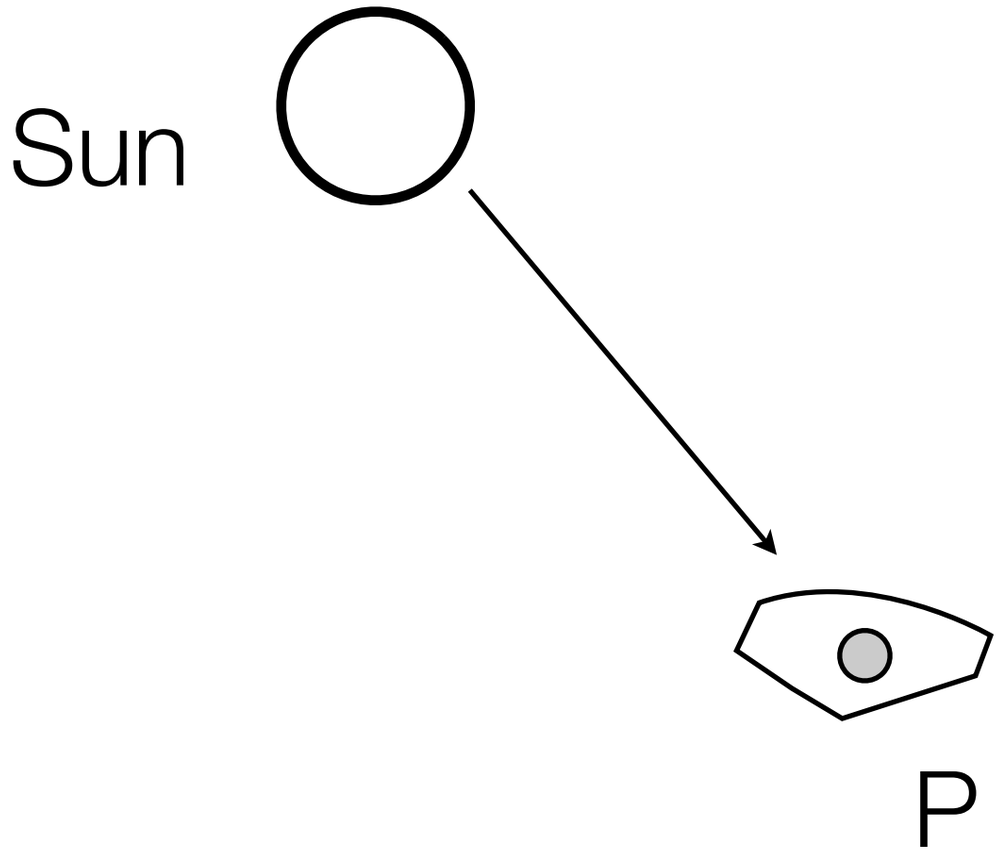


P

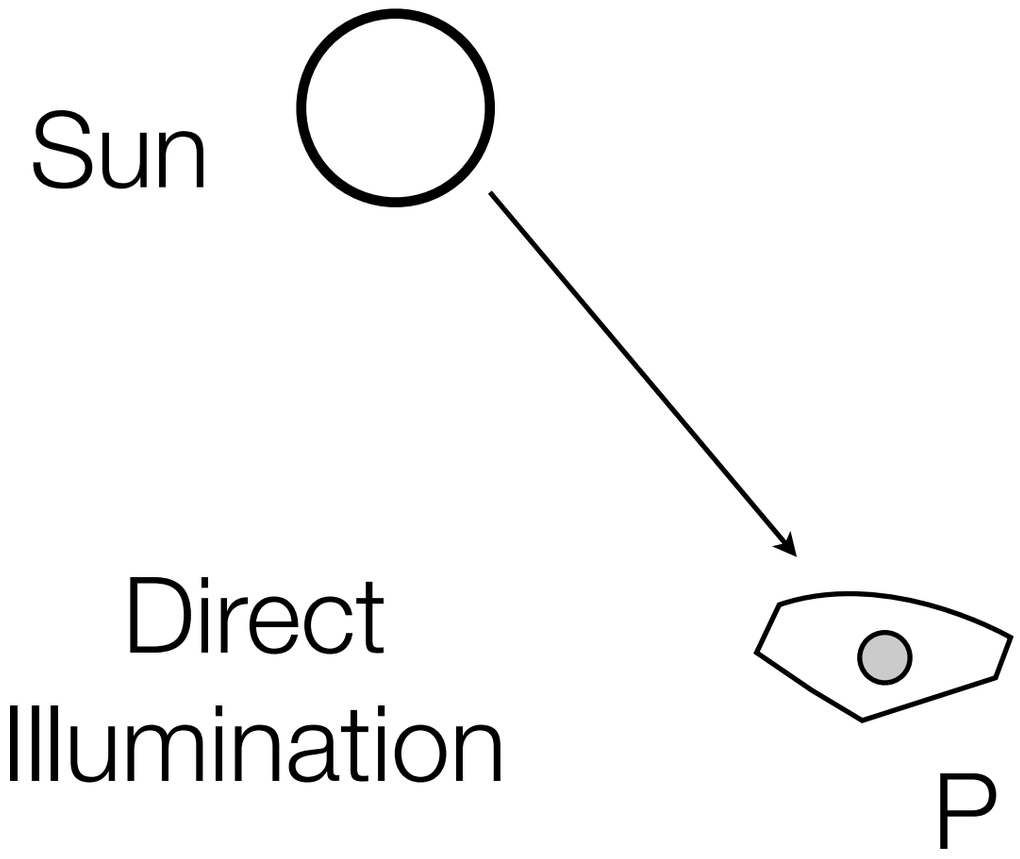
Illumination in Computer Graphics



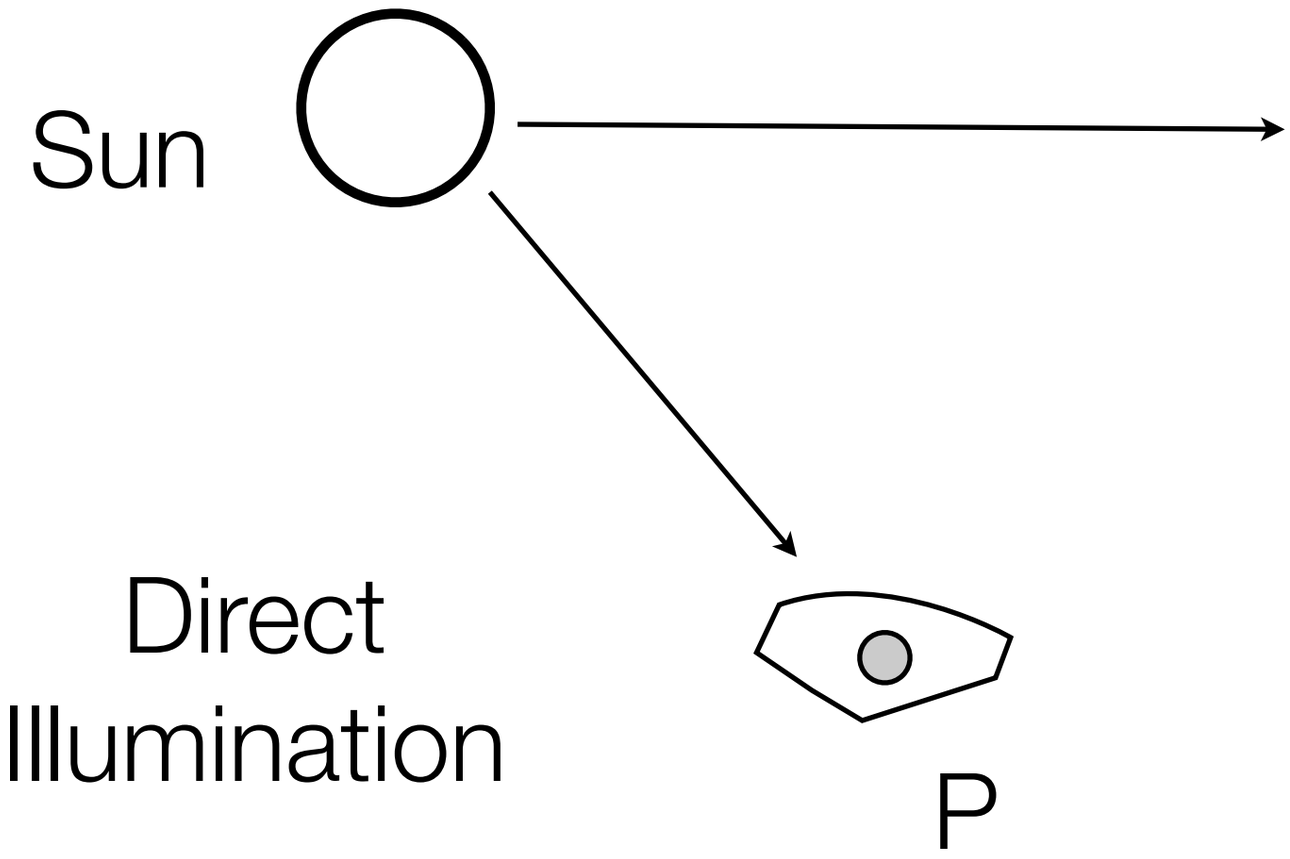
Illumination in Computer Graphics



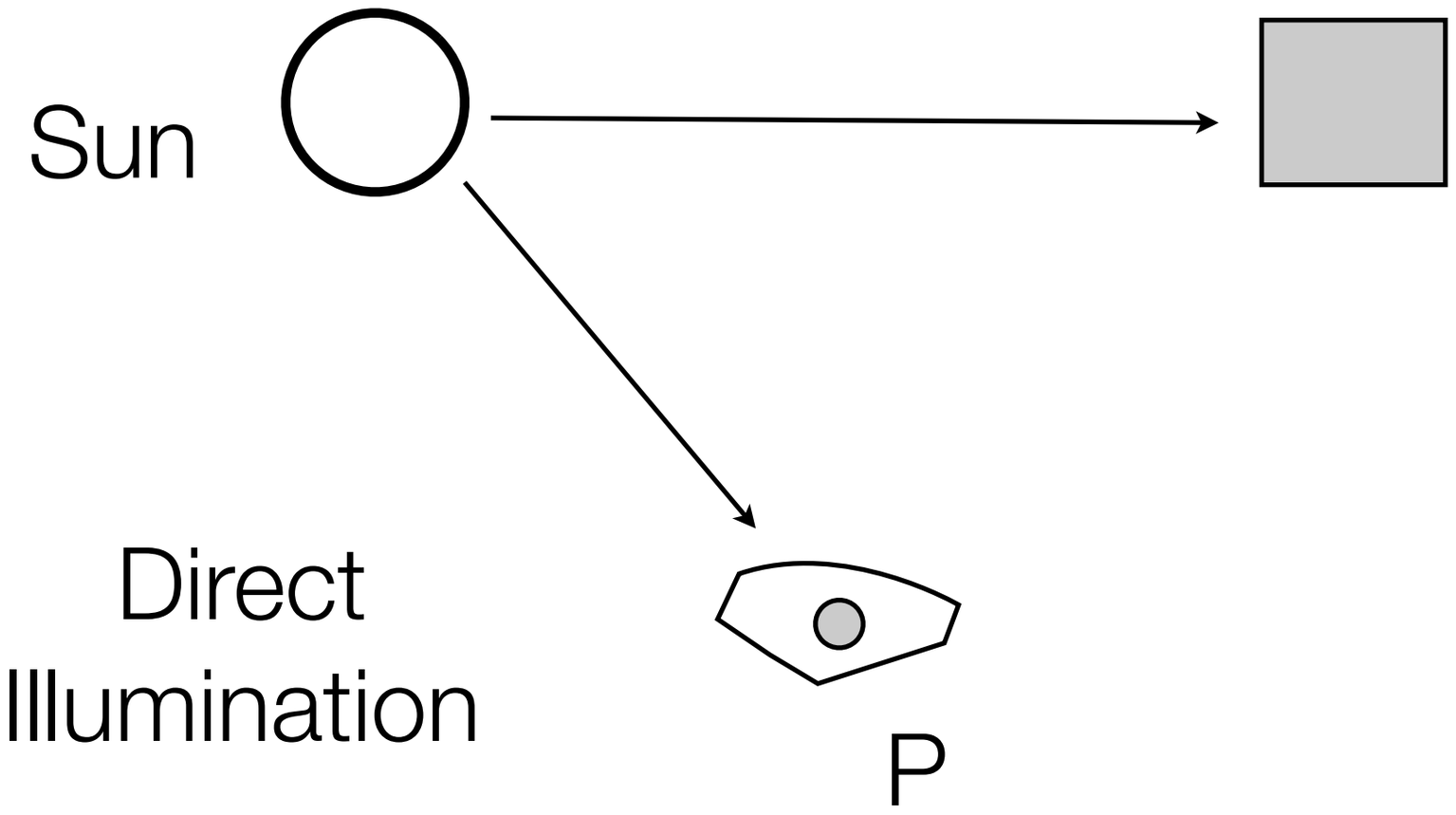
Illumination in Computer Graphics



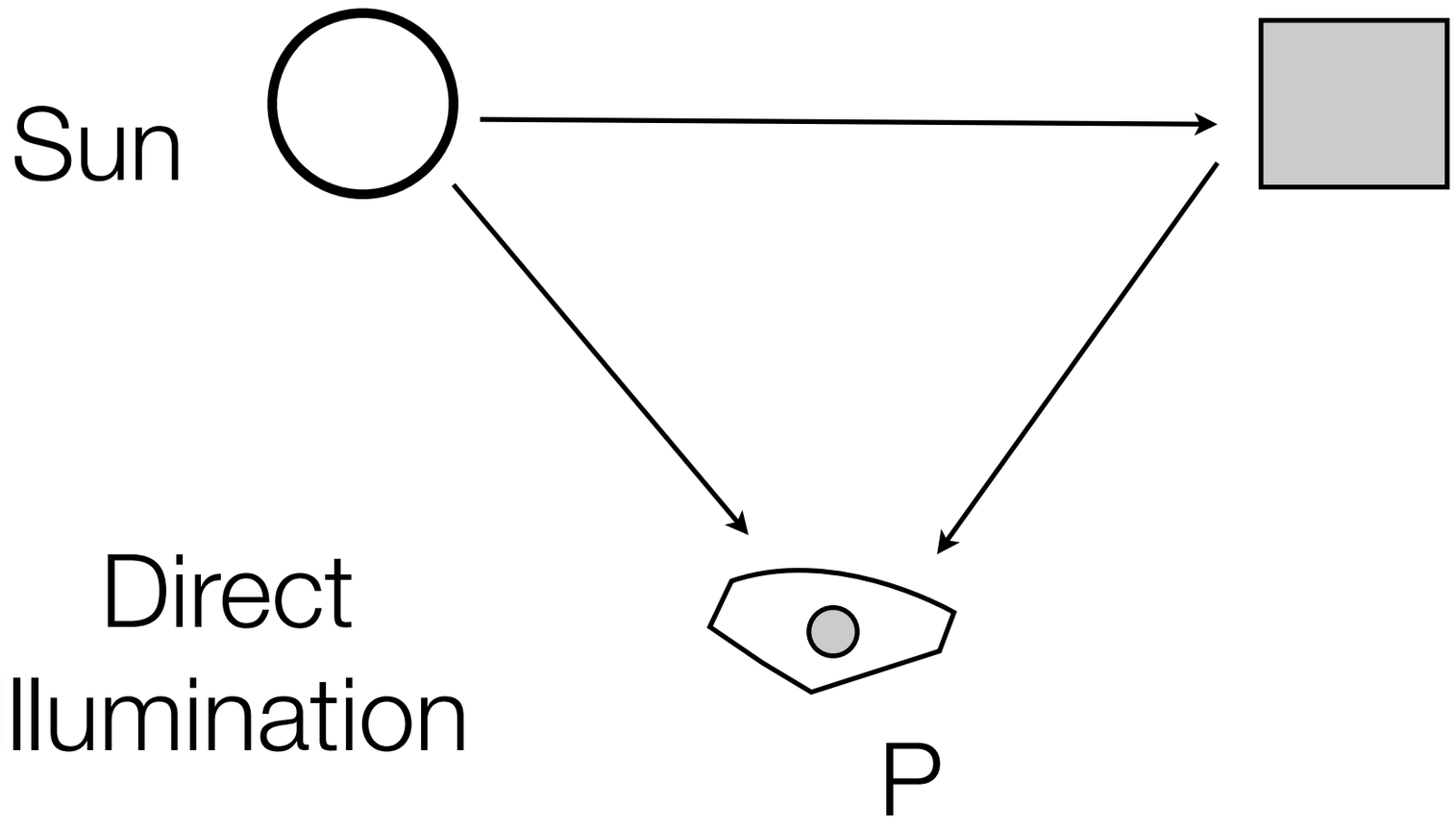
Illumination in Computer Graphics



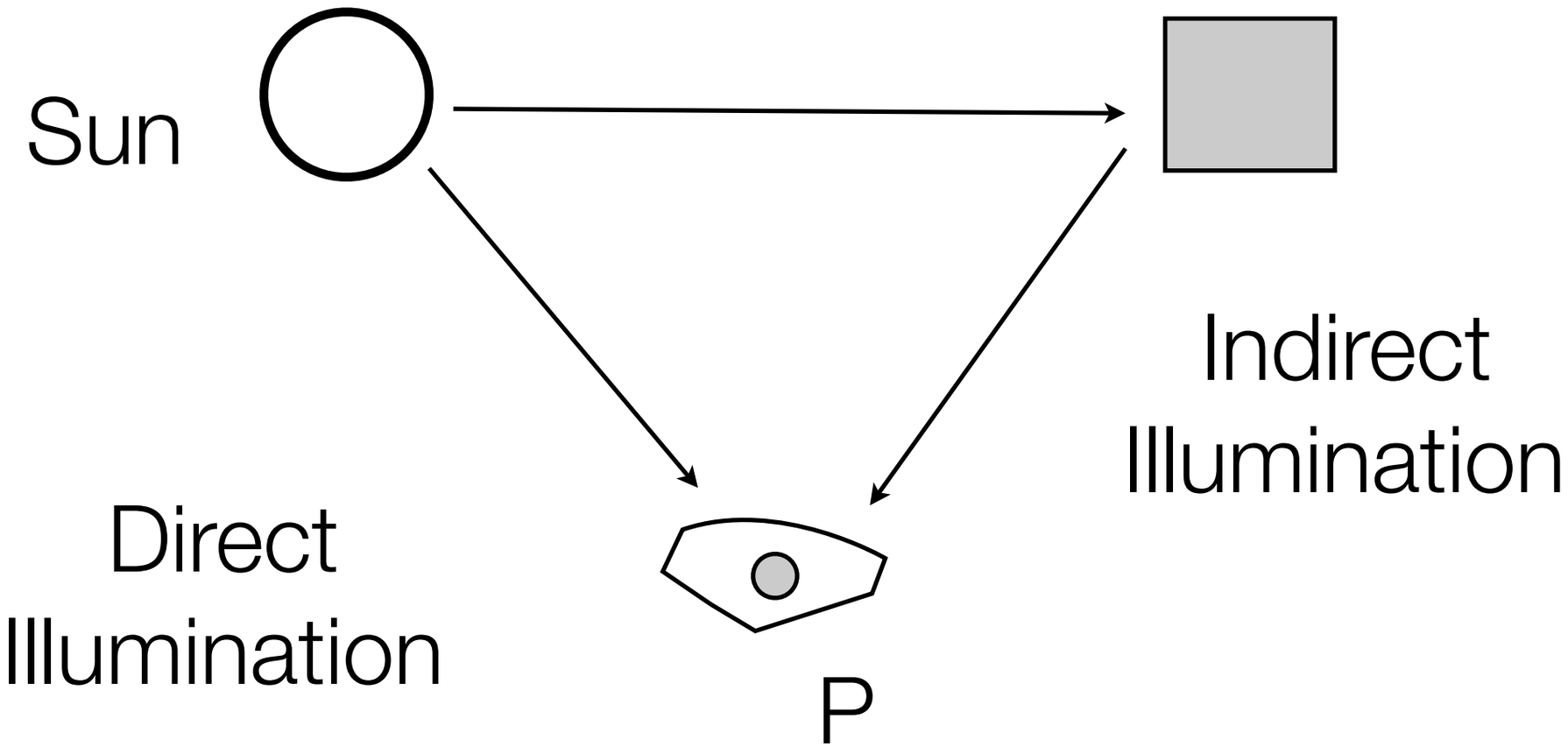
Illumination in Computer Graphics



Illumination in Computer Graphics



Illumination in Computer Graphics



“Global” Illumination

- What is “global”?
  - The light arrives at a point from all sources
  - The light-surface interaction is “global” in nature
- **Goal:** Make the renderings as photo-realistic as possible
- **Issues:**
  - How to track light as it travels through the scene?
  - What about objects that are not self-illuminated?

Side-effects of Global Illumination

- Object-light interaction is more natural
- No special case processing for shadow generation
  - Shadows are regions where there is “not enough” incident illumination available
- Vigorous research problem in its own right

Models of Global Illumination

Mathematical Model

- Model proposed by Kajiya (1986)
  - "... to provide a unified context for viewing rendering algorithms as more or less accurate approximations to the solutions for a single equation."
- Completely general mathematical statement
- Not tied down to any particular assumption about the light transport behavior

Kajiya's Equation

$$I(x,x') = g(x,x')[\xi(x,x') + \int \rho(x,x',x'')I(x',x'')dx'']$$

Kajiya's Equation

$$I(x, x') = g(x, x')[\xi(x, x') + \int \rho(x, x', x'')I(x', x'')dx'']$$

$I(x, x')$: transport intensity or the intensity of light passing from point x' to point x

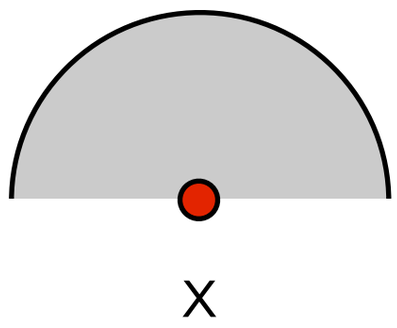
$g(x, x')$: visibility function between x and x'. If x and x' cannot 'see' each other, then this value is zero, else g varies as the square of the distance between them.

$\xi(x, x')$: the transfer emittance from x' to x related to any self-emitted light by x' towards x

$\rho(x, x', x'')$: is the scattering term wrt. direction x' and x''. It is the intensity of the energy scattered towards x from the point on surface x' arriving from point x''. Also known as the "unoccluded three-point transport reflectance."

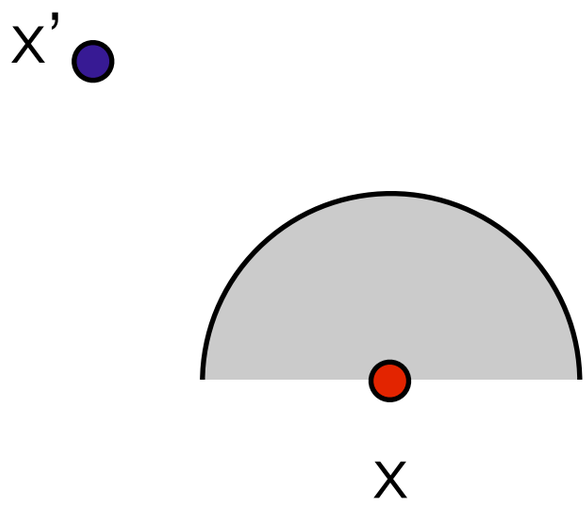
Integral Term in Kajiya's Equation

- Integral is defined over all points on all surfaces in the scene
- Equivalently - all points surrounding point x
  - If we consider only front-faces of polygons, the region simplifies to an hemisphere around x



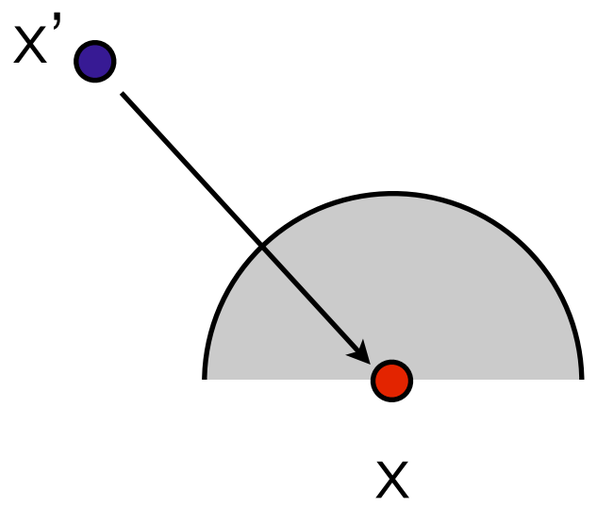
Integral Term in Kajiya's Equation

- Integral is defined over all points on all surfaces in the scene
- Equivalently - all points surrounding point x
  - If we consider only front-faces of polygons, the region simplifies to an hemisphere around x



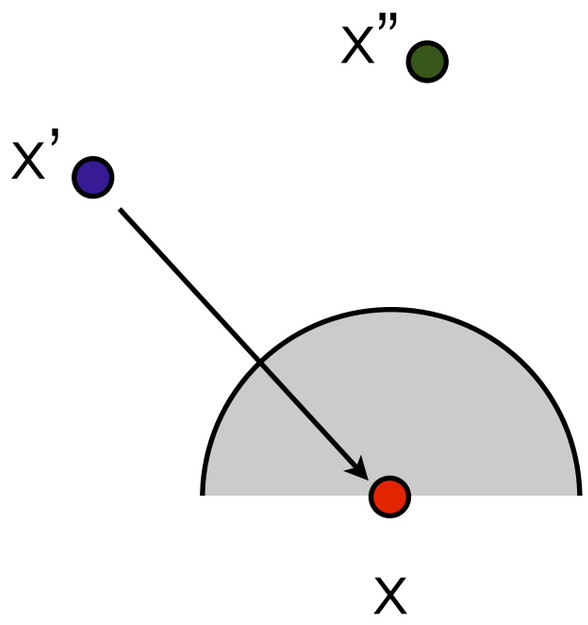
Integral Term in Kajiya's Equation

- Integral is defined over all points on all surfaces in the scene
- Equivalently - all points surrounding point x
  - If we consider only front-faces of polygons, the region simplifies to an hemisphere around x



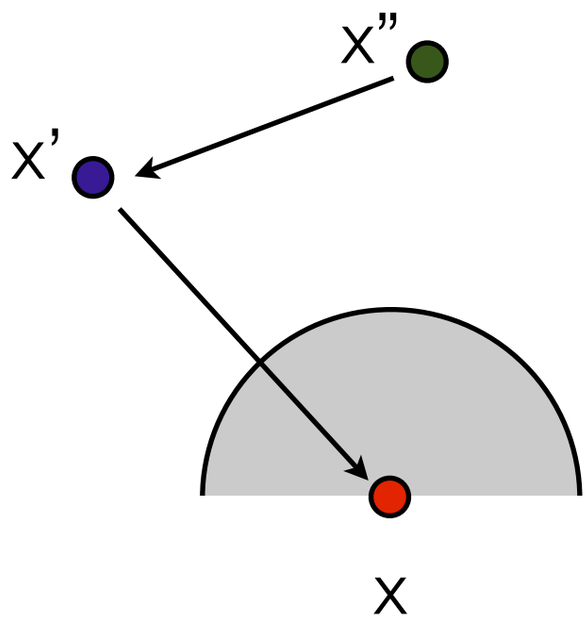
Integral Term in Kajiya's Equation

- Integral is defined over all points on all surfaces in the scene
- Equivalently - all points surrounding point x
  - If we consider only front-faces of polygons, the region simplifies to an hemisphere around x



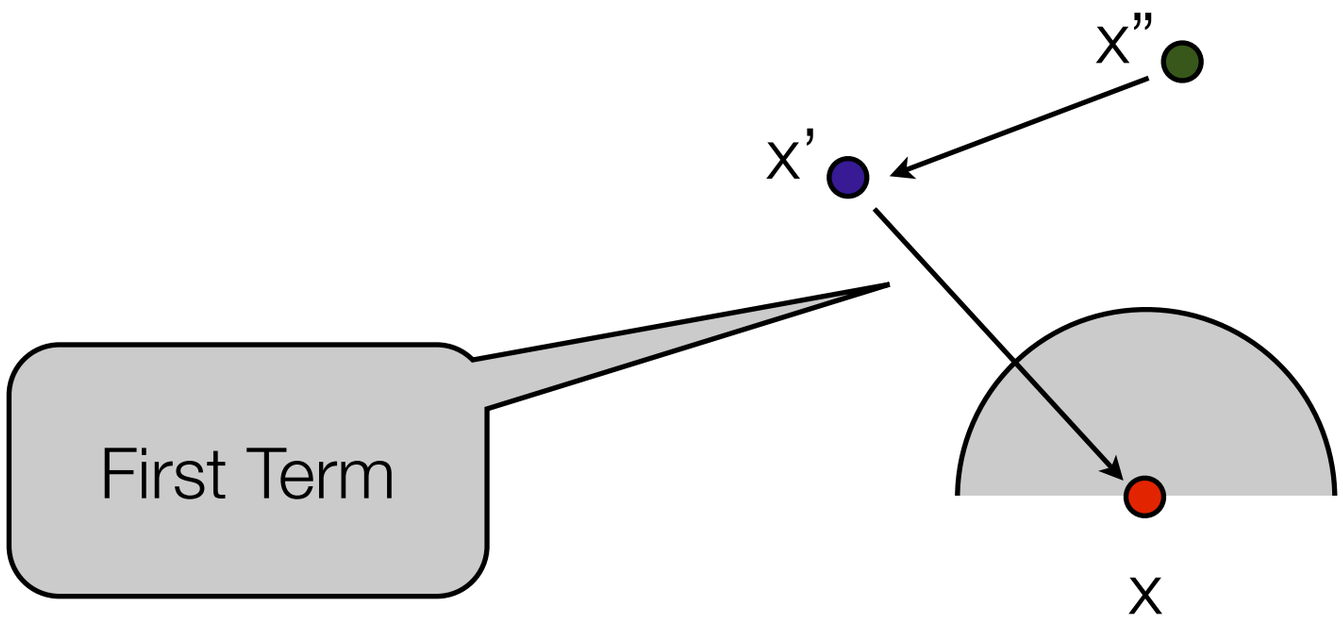
Integral Term in Kajiya's Equation

- Integral is defined over all points on all surfaces in the scene
- Equivalently - all points surrounding point x
  - If we consider only front-faces of polygons, the region simplifies to an hemisphere around x



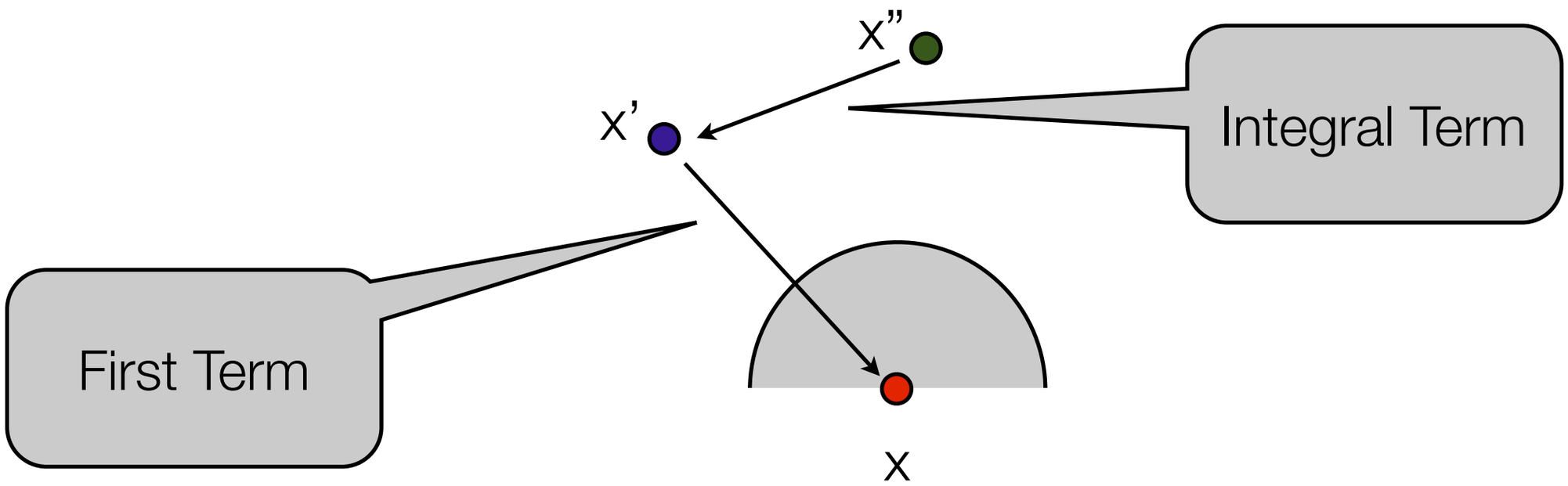
Integral Term in Kajiya's Equation

- Integral is defined over all points on all surfaces in the scene
- Equivalently - all points surrounding point x
  - If we consider only front-faces of polygons, the region simplifies to an hemisphere around x



Integral Term in Kajiya's Equation

- Integral is defined over all points on all surfaces in the scene
- Equivalently - all points surrounding point x
  - If we consider only front-faces of polygons, the region simplifies to an hemisphere around x



Observations

Observations

- The integral cannot be evaluated analytically in a reasonably real-time fashion
  - Reduce complexity by taking an “approximate” approach
  - Direct evaluation done using Monte Carlo methods

Observations

- The integral cannot be evaluated analytically in a reasonably real-time fashion
  - Reduce complexity by taking an “approximate” approach
  - Direct evaluation done using Monte Carlo methods
- The equation is dependent on the location of point x in the scene
  - In fact, point x is every point in the scene !
  - View-dependent representation - can help in reducing the complexity of the solution

Observations

- The integral cannot be evaluated analytically in a reasonably real-time fashion
  - Reduce complexity by taking an “approximate” approach
  - Direct evaluation done using Monte Carlo methods
- The equation is dependent on the location of point x in the scene
  - In fact, point x is every point in the scene !
  - View-dependent representation - can help in reducing the complexity of the solution
- Recursive equation
  - Find I(x’,x”) to calculate I(x,x’)
  - Leads to an elegant **path tracing** algorithm for light transport

Implementing Kajiya's Equation

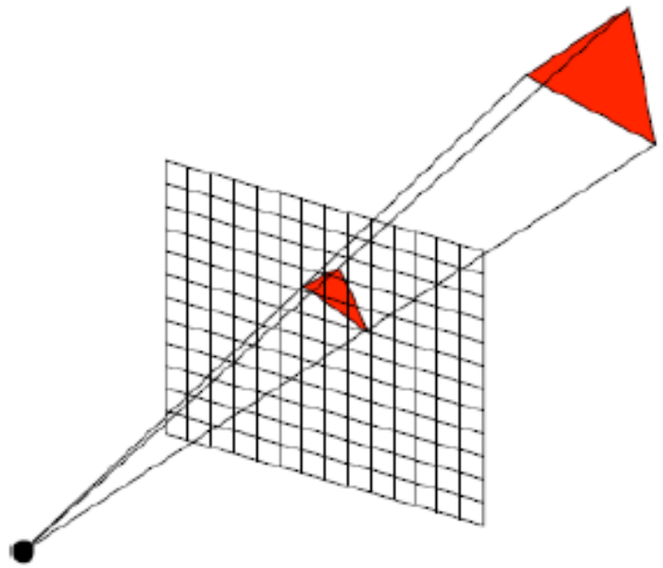
- **Rendering** : visualizing 3D data on 2D screen
- Two main approaches
  - Projective Methods
  - Path Tracing Methods

Projective Vs Ray Tracing

Projective methods:

Object-order rendering, i.e.

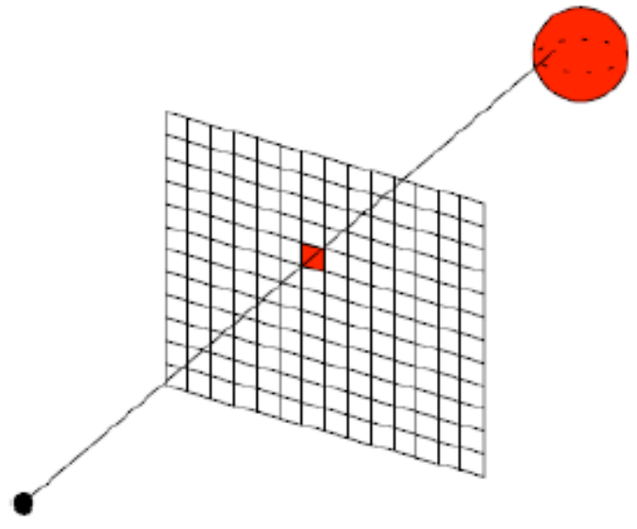
- For each object . . .
- . . . find and update all pixels that it influences



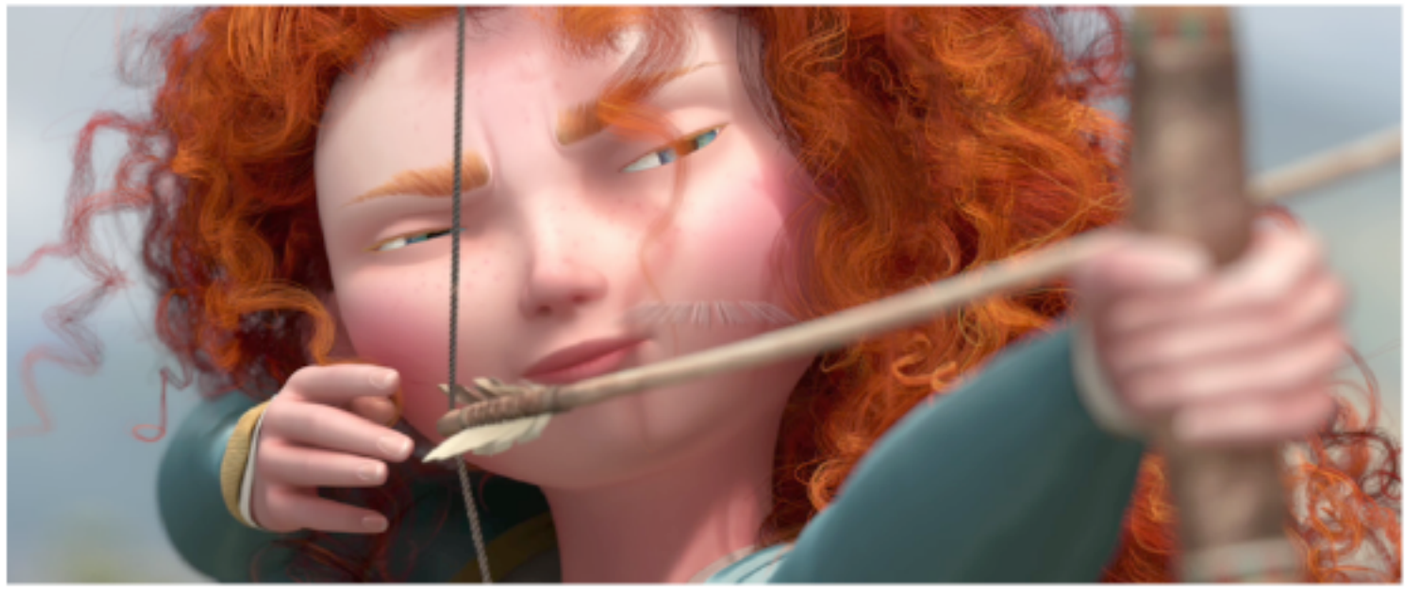
Ray tracing:

Image-order rendering, i.e.

- For each pixel . . .
- . . . find all objects that influence it and update it accordingly



Examples : Ray Traced Rendering

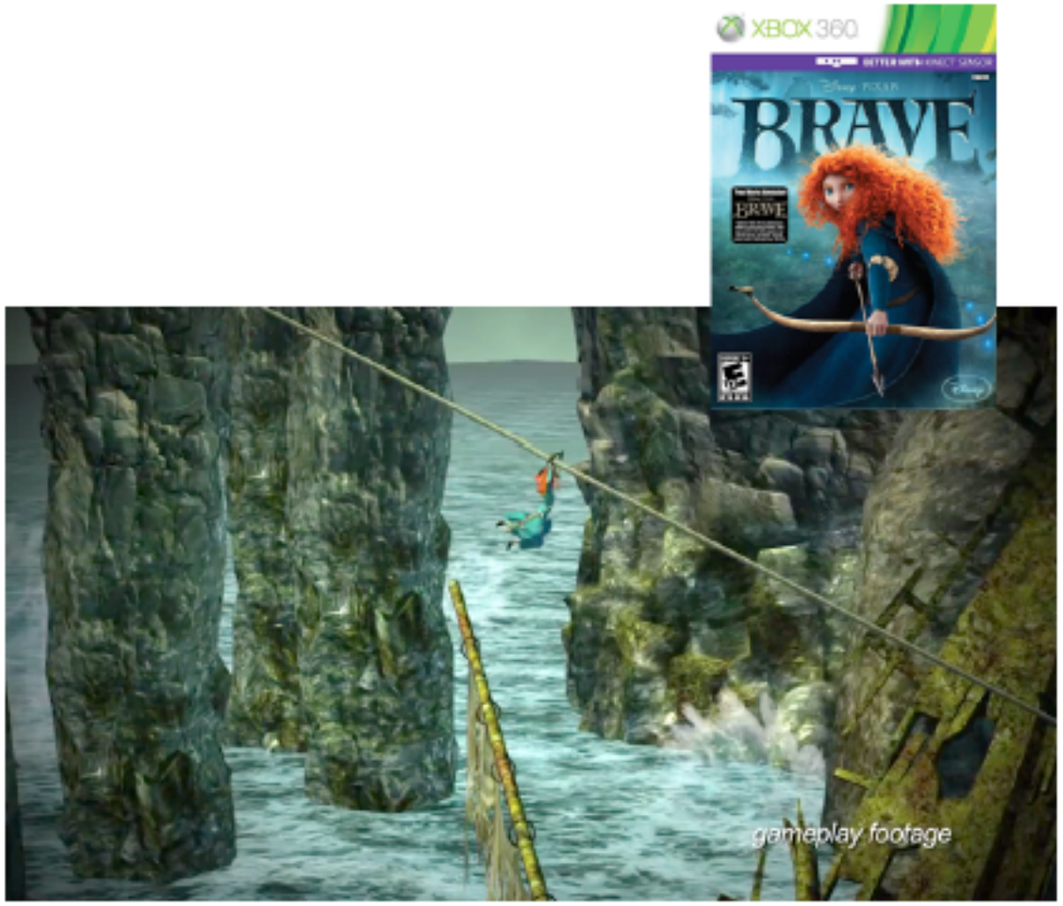


Images courtesy Wolfgang Hürst, Utrecht University

Example : (Real-time) Projected Rendering

Images courtesy Wolfgang Hürst, Utrecht University

Examples : Comparison



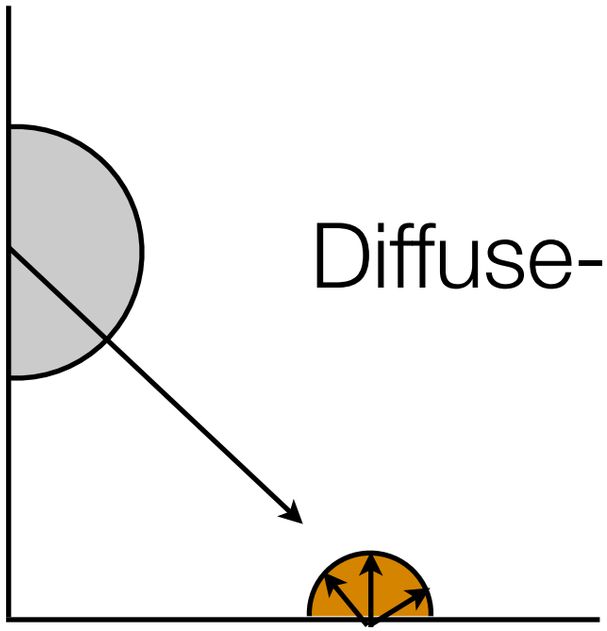
Images courtesy Wolfgang Hürst, Utrecht University

Surface-Light Transport Model

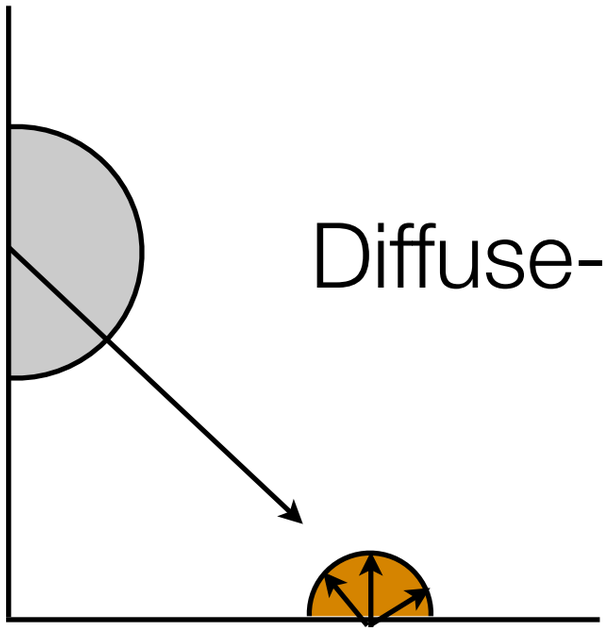
- Categorized based on the surface to surface interactions
- Non-mathematical organization, enables easy comparison between the various algorithms
- One surface (or point) is considered the “source” and another is the “sensor”
- What types of light transport occurs between the source and the sensor
  - Diffuse light - isotropic - same in all directions
  - Specular light - anisotropic - directional behavior

Light Transport Mechanisms

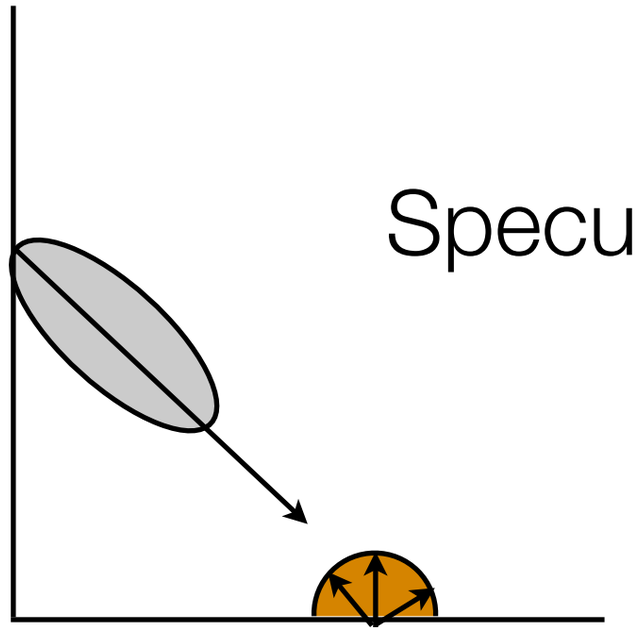
Light Transport Mechanisms



Light Transport Mechanisms

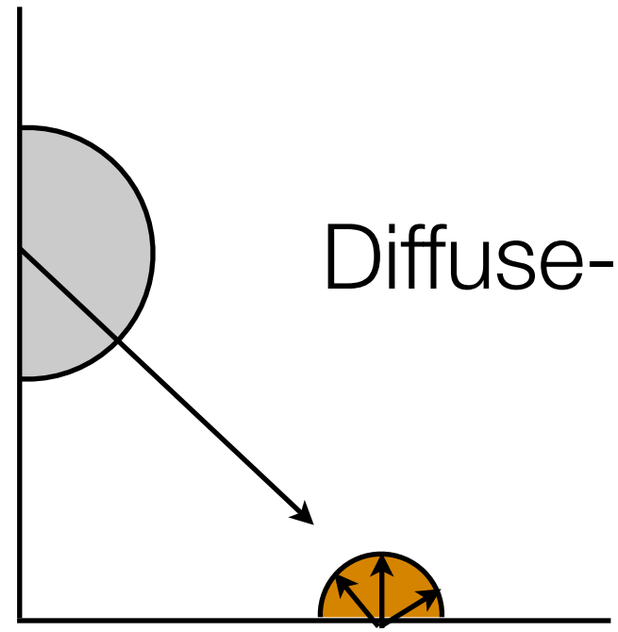


Diffuse-Diffuse

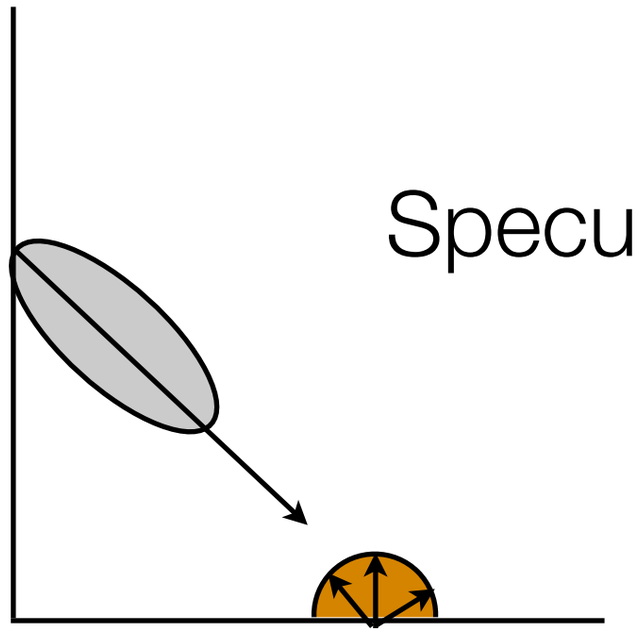


Specular-Diffuse

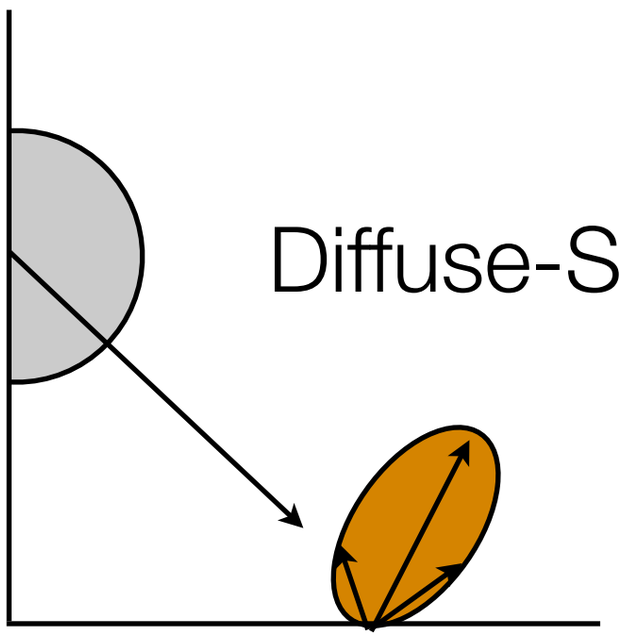
Light Transport Mechanisms



Diffuse-Diffuse

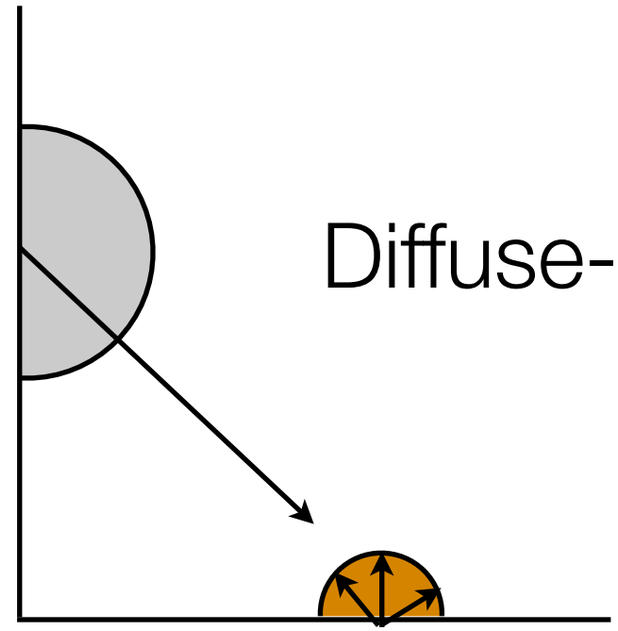


Specular-Diffuse

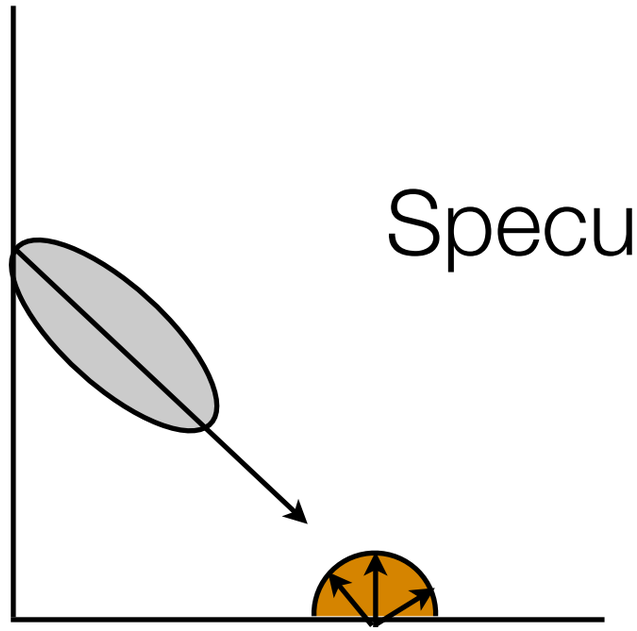


Diffuse-Specular

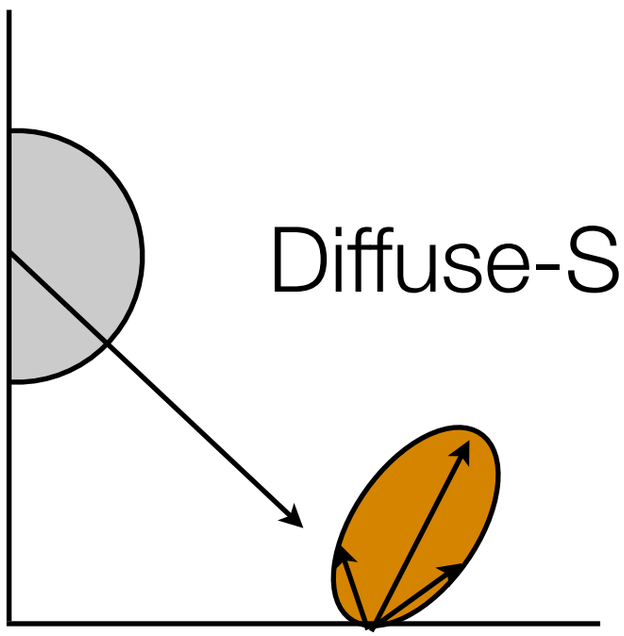
Light Transport Mechanisms



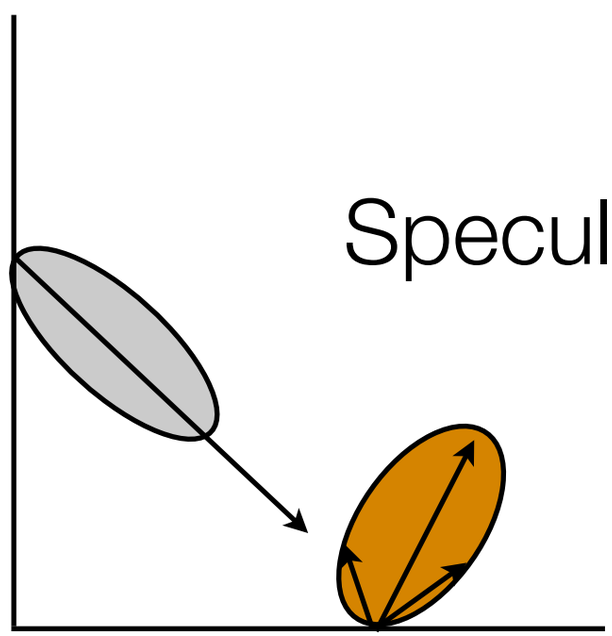
Diffuse-Diffuse



Specular-Diffuse



Diffuse-Specular



Specular-Specular

Methods to Solve Light Transport Models

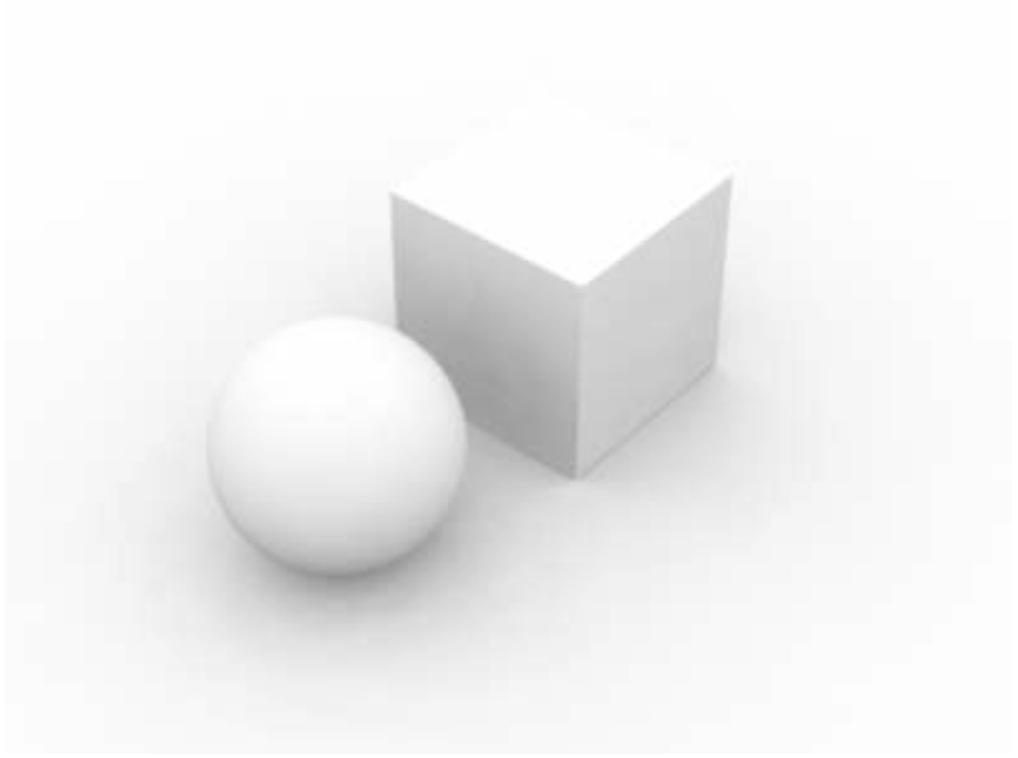
- Diffuse - Diffuse
  - Only diffuse surfaces exist in the scene (Matte finish)
  - Radiosity methods used to “solve” for the equation
- Specular - Specular, Specular - Diffuse
  - Shiny surfaces with reflective properties
  - Ray-tracing methods

Simplifying the Light Contribution

- Three components
  - Ambient
  - Diffuse
  - Specular
- Ambient component is assumed to be a constant value in most local illumination models
  - Not observed in practice

Simplifying the Light Contribution

- Three components
  - Ambient
  - Diffuse
  - Specular
- Ambient component is assumed to be a constant value in most local illumination models
  - Not observed in practice



Global Vs Local Diffuse-Diffuse Interaction

Direct lighting

Global lighting

Dynamic Ambient Component

- Ambient term
  - The irradiance that reaches a point (x) as the summed contribution of the diffuse and specular emissions from all other points in the scene
- General effect - occluded parts of the scene are darker than the exposed portions
- Ambient occlusion (AO) computed by ray-tracing approaches
  - Very slow

Screen Space Ambient Occlusion

Motivation

- Accurate AO calculation is expensive
- Basic idea - Areas that are obscured by other objects are darker than those that are not
- Manipulation of intensity levels
- Can be approximated with screen space methods

Is SSAO necessary?

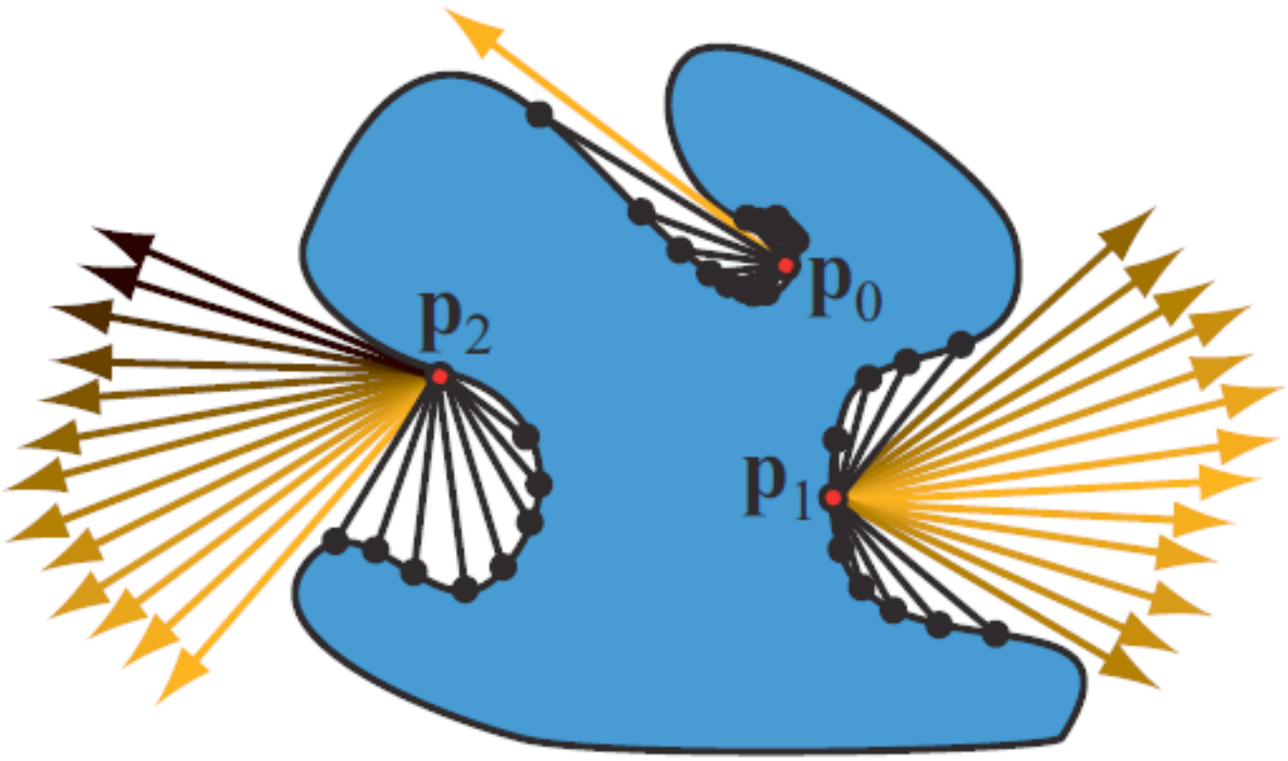
Is SSAO necessary?

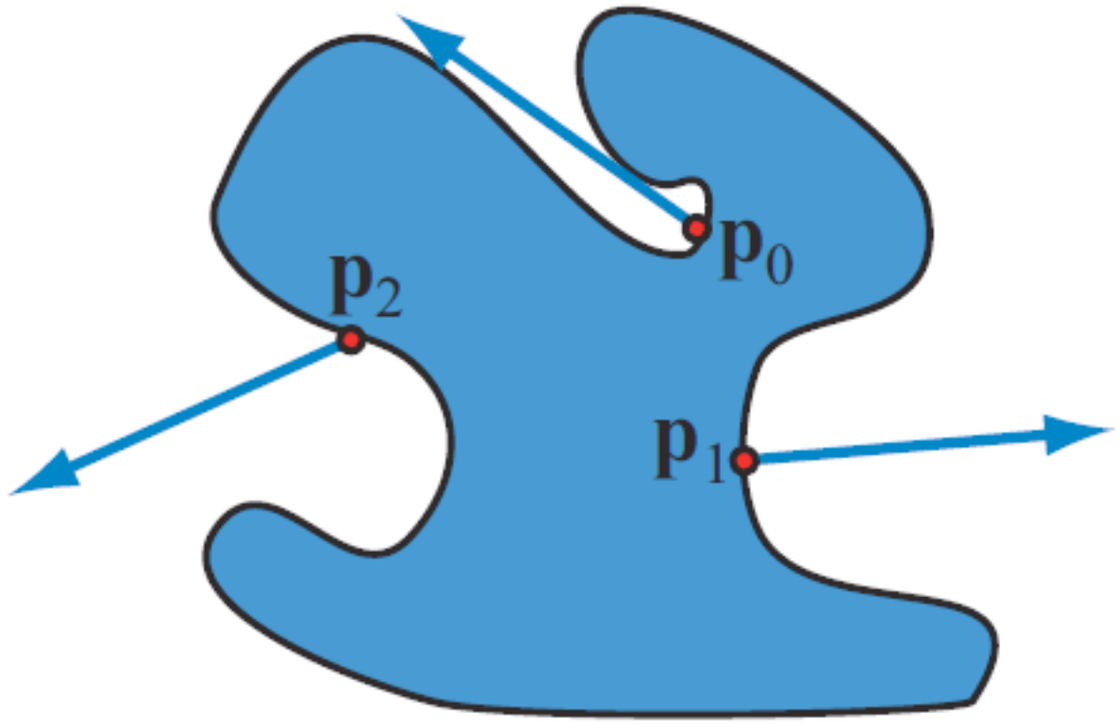
Is SSAO necessary?

Ambient Occlusion - Ray Cast Solution

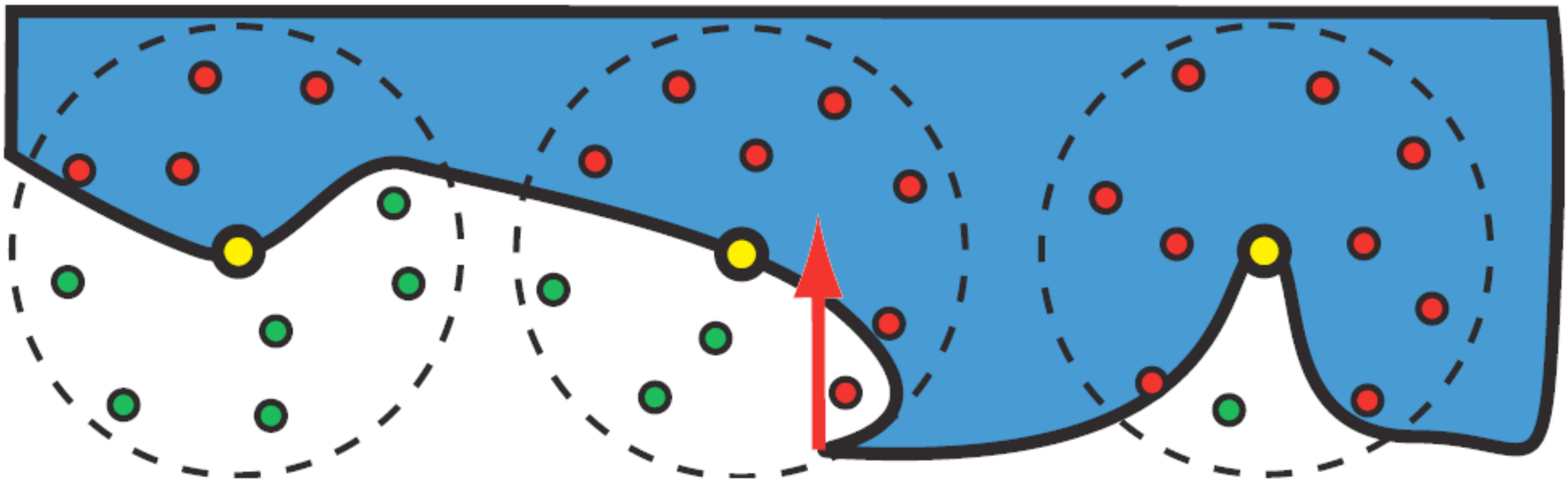
- For every point on the surface
  - Cast a set of rays in all directions surrounding the point
  - Count how many rays continue unobstructed towards the viewer
    - Can be replaced by a threshold distance
  - Use the proportion of obstructed rays as a measure of “occlusion” from surrounding surfaces

Computing Occlusion

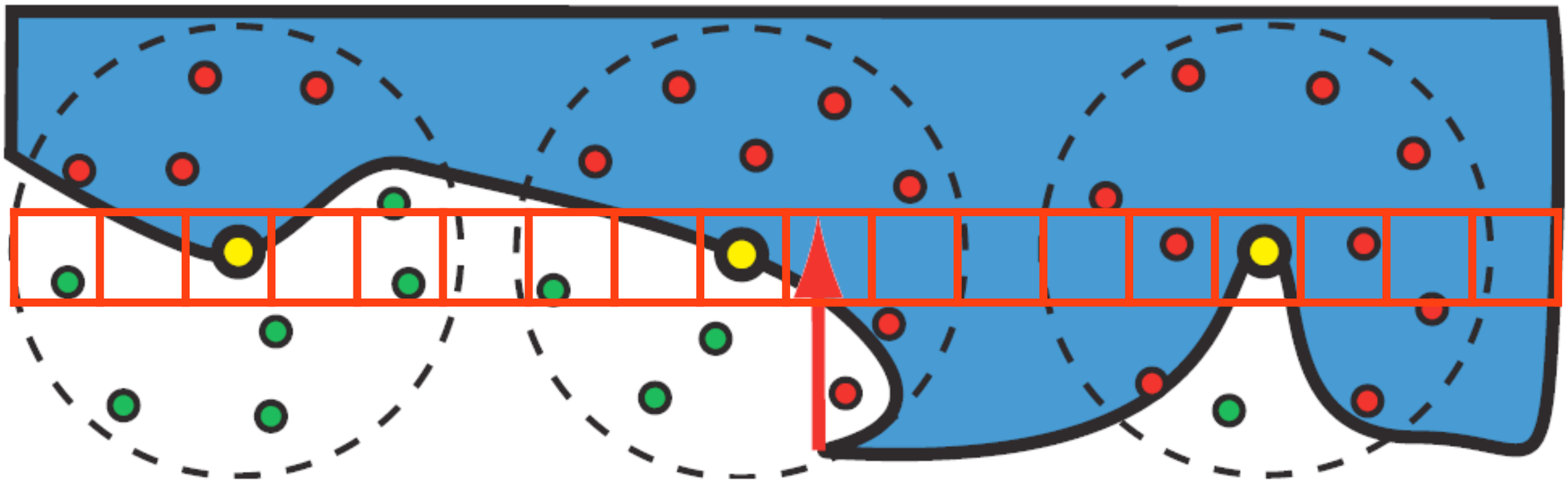




Screen Space Ambient Occlusion
Overall Idea



Screen Space Ambient Occlusion
Overall Idea



Overall Idea

Overall Idea

- Store the depth value for each fragment

Overall Idea

- Store the depth value for each fragment
- Check within a region around the current location

Overall Idea

- Store the depth value for each fragment
- Check within a region around the current location
  - Generate random samples around the 3D point

Overall Idea

- Store the depth value for each fragment
- Check within a region around the current location
  - Generate random samples around the 3D point
  - Find the depth value at these neighboring locations

Overall Idea

- Store the depth value for each fragment
- Check within a region around the current location
  - Generate random samples around the 3D point
  - Find the depth value at these neighboring locations
  - If more than 50% of the values are LESS than the current fragment depth value

Overall Idea

- Store the depth value for each fragment
- Check within a region around the current location
  - Generate random samples around the 3D point
  - Find the depth value at these neighboring locations
- If more than 50% of the values are LESS than the current fragment depth value
  - More than 50% of the neighboring fragments are closer to the camera than the current fragment

Overall Idea

- Store the depth value for each fragment
- Check within a region around the current location
  - Generate random samples around the 3D point
  - Find the depth value at these neighboring locations
- If more than 50% of the values are LESS than the current fragment depth value
  - More than 50% of the neighboring fragments are closer to the camera than the current fragment
  - The current fragment is probably in shadow region!

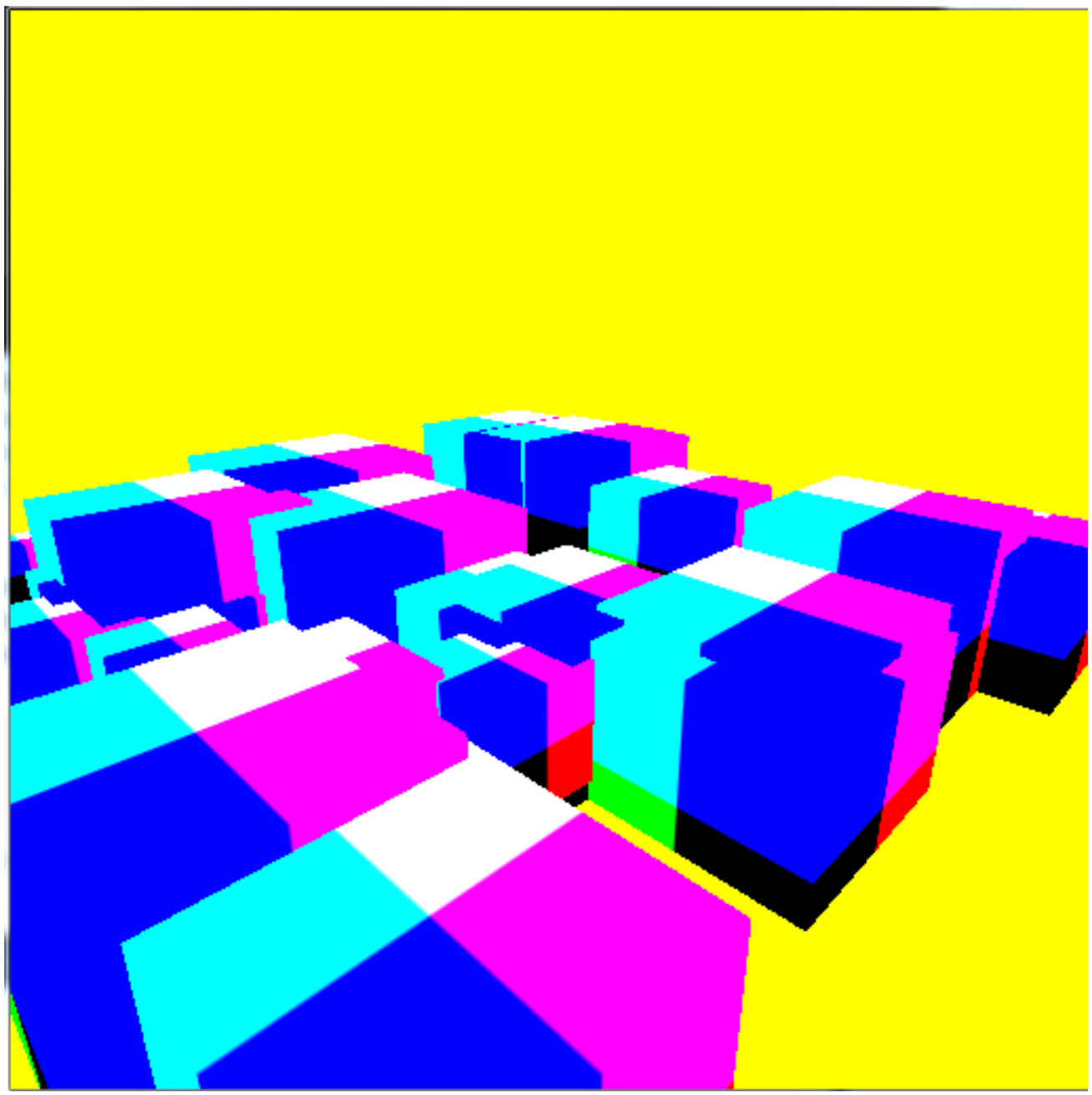
Overall Idea

- Store the depth value for each fragment
- Check within a region around the current location
  - Generate random samples around the 3D point
  - Find the depth value at these neighboring locations
- If more than 50% of the values are LESS than the current fragment depth value
  - More than 50% of the neighboring fragments are closer to the camera than the current fragment
  - The current fragment is probably in shadow region!
  - Compute a “blocking factor” to attenuate the light values

Algorithm

- Multi-pass implementation
- Needs render-to-texture functionality
- First pass
  - Render the 3D scene
- Subsequent passes
  - Render a fullscreen quad to process the fragments

Example Scene



3D Cubes

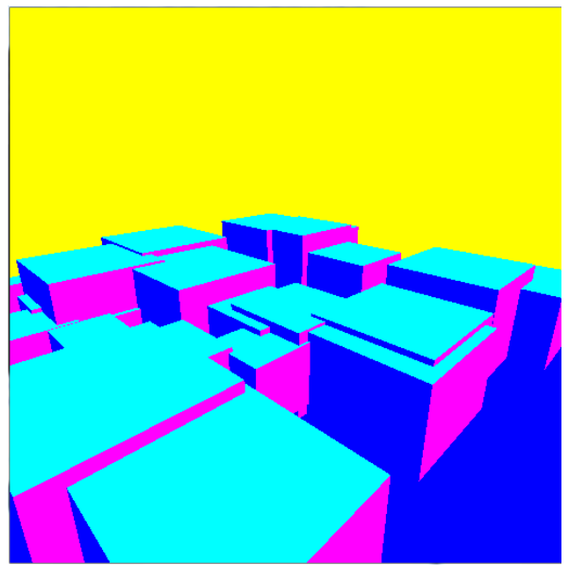
Color = local coordinate values

Pass 1 (Fragment Shader)

- Inputs
  - Camera position
  - Eye space position
  - Eye space normals
  - Projection matrix
  - 7x7 matrix of random values

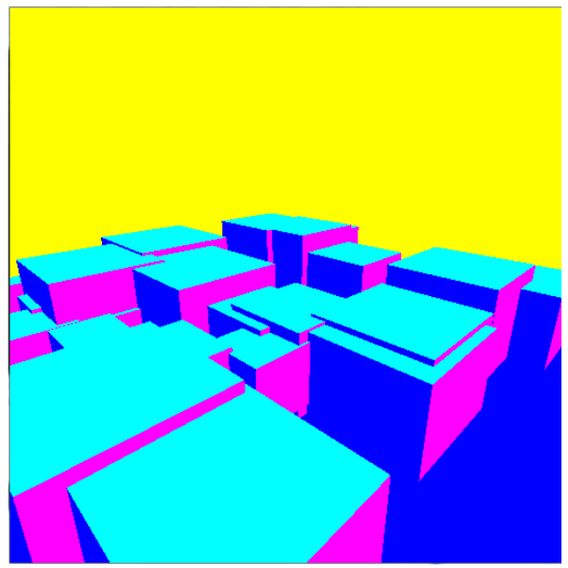
Pass 1 (Fragment Shader)

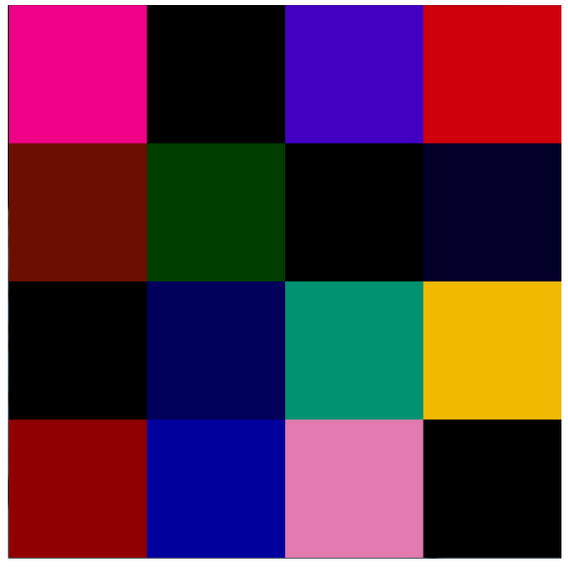
- Inputs
  - Camera position
  - Eye space position
  - Eye space normals
  - Projection matrix
  - 7x7 matrix of random values



Pass 1 (Fragment Shader)

- Inputs
  - Camera position
  - Eye space position
  - Eye space normals
  - Projection matrix
  - 7x7 matrix of random values





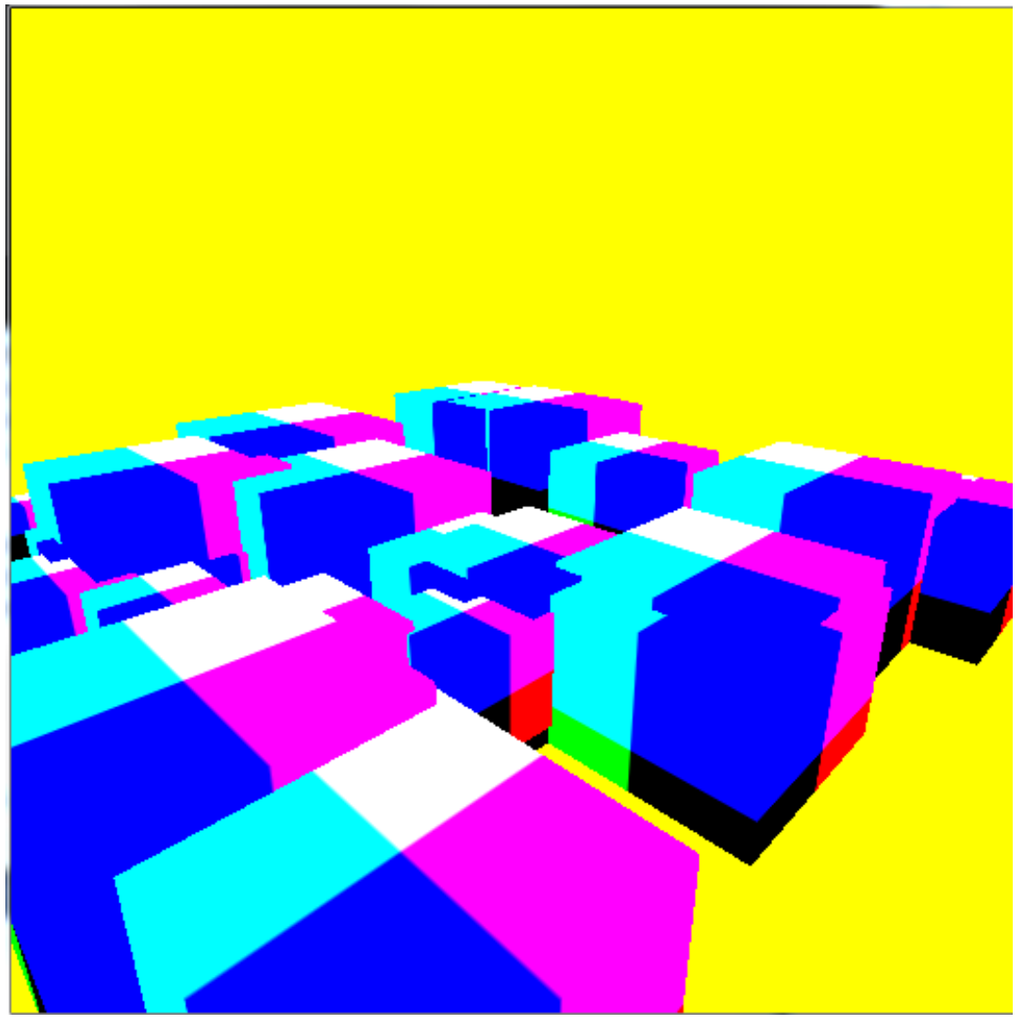
Calculating the “Blocking Factor”

- Generate random offset vectors in a sphere, around the point under consideration
- Compare the depth values for the fragment and the neighbor at the random offset
- Calculate the blocking factor based on the depth comparison
- (In supplied code) blocking factor is calculated as a cumulative function of the z-depth difference between the point and its neighbors

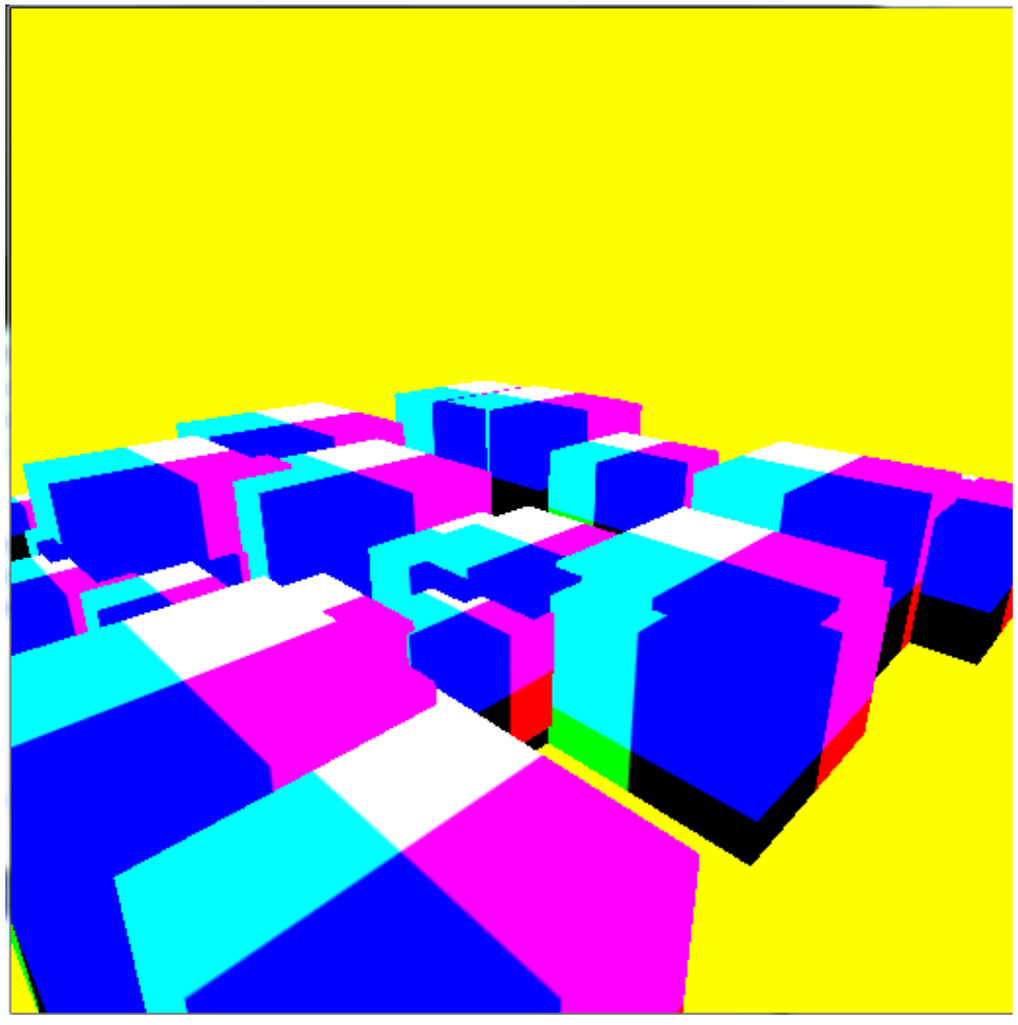
GLSL Code Snippet

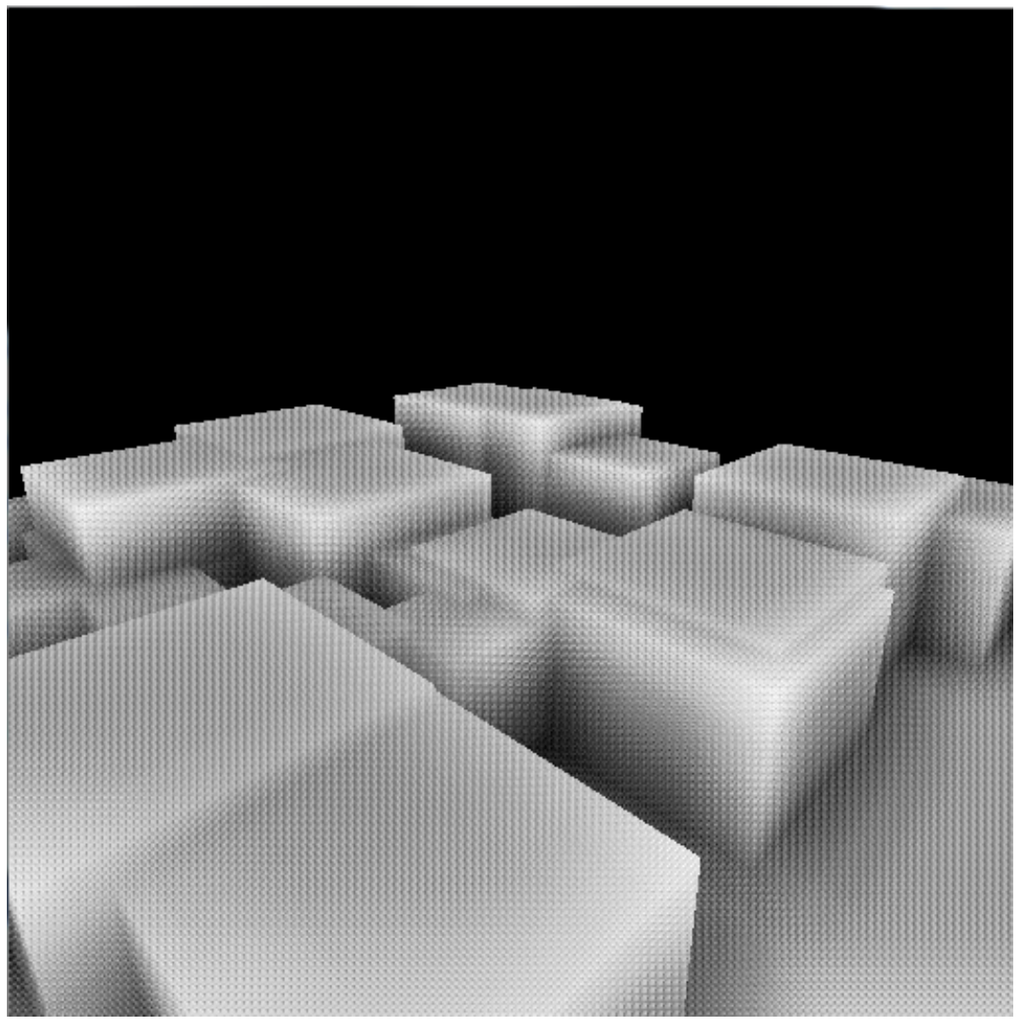
```
// for each sample
    int count = 0;
    for( int samplex = 0; samplex < 7; ++samplex )
    {
            for( int sampley = 0; sampley < 7; ++ sampley )
            {
                vec2 index = vec2( samplex, sampley );
                vec4 rand_lookup = textureRect( rand_values, index );
                vec4 rotated_vector = rotationMatrix * rand_lookup;
                vec4 offset = rotated_vector * radius;
                newPos3d = pos3d + offset;

                vec4 projectedPoint = projectionMatrix * vec4(newPos3d.x, newPos3d.y, newPos3d.z, 1.0);
                vec2 screenPoint =  projectedPoint.xy / projectedPoint.w;

                vec2 depth_lookup = (screenPoint.xy*0.5) + vec2(0.5, 0.5);
                vec3 neighbor_lookup = textureRect( eyeSpacePosition, depth_lookup.xy*512 ).xyz;
                neighbor_lookup.z = -1.0/neighbor_lookup.z;

                 float z_diff = 50.0 * max( 0.0, newPos3d.z - neighbor_lookup.z );
                 block_factor += 1.0 / ( 1.0 + (z_diff*z_diff) );
            }
 }

block_factor /= 49;
block_factor = 1.0 - block_factor;
```

Visualizing the Blocking Factor

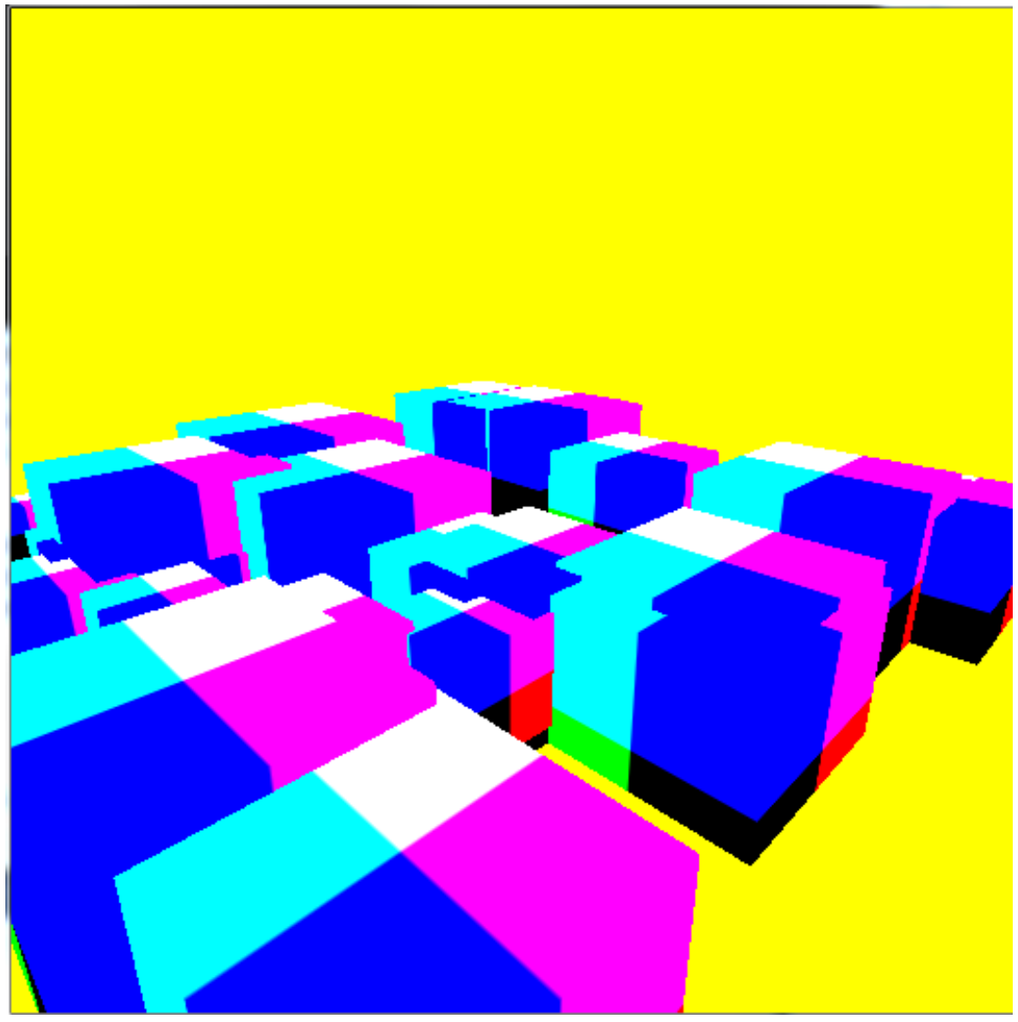


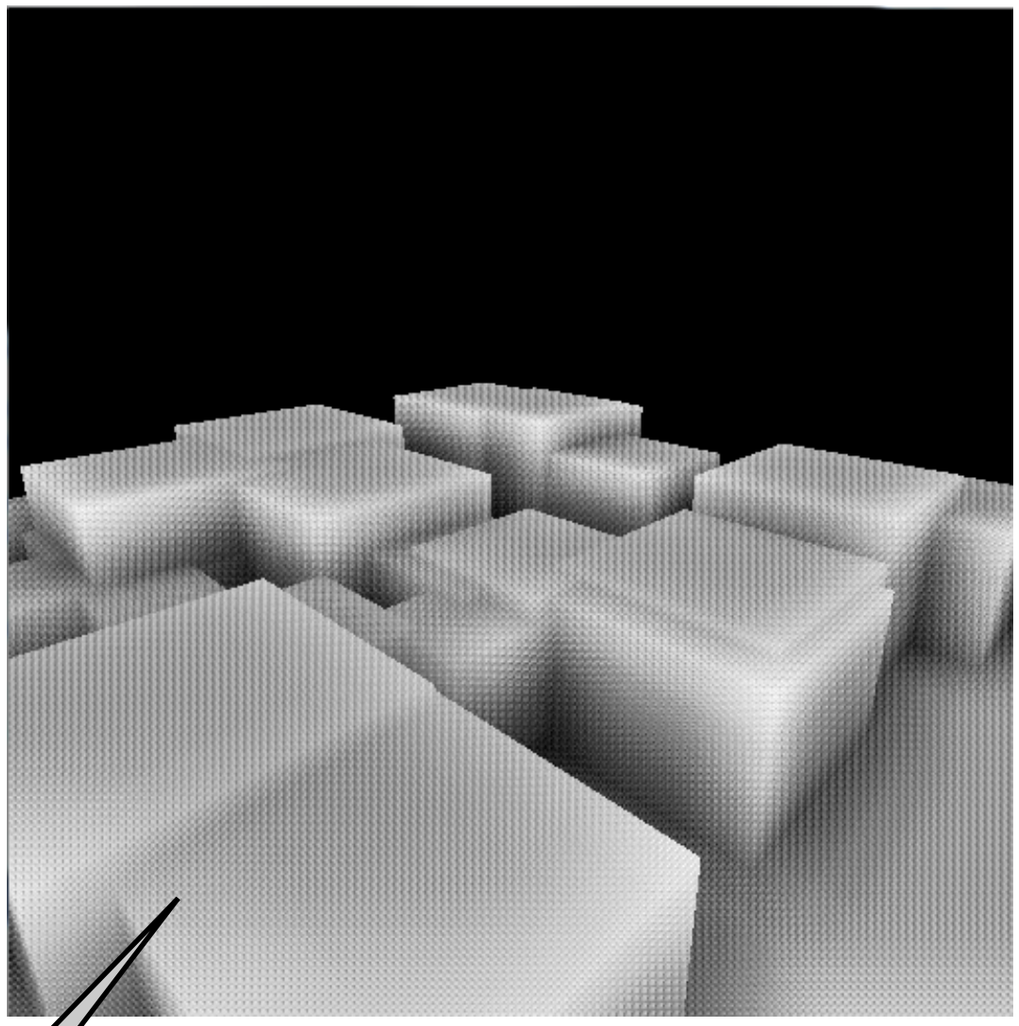
Visualizing the Blocking Factor





Visualizing the Blocking Factor

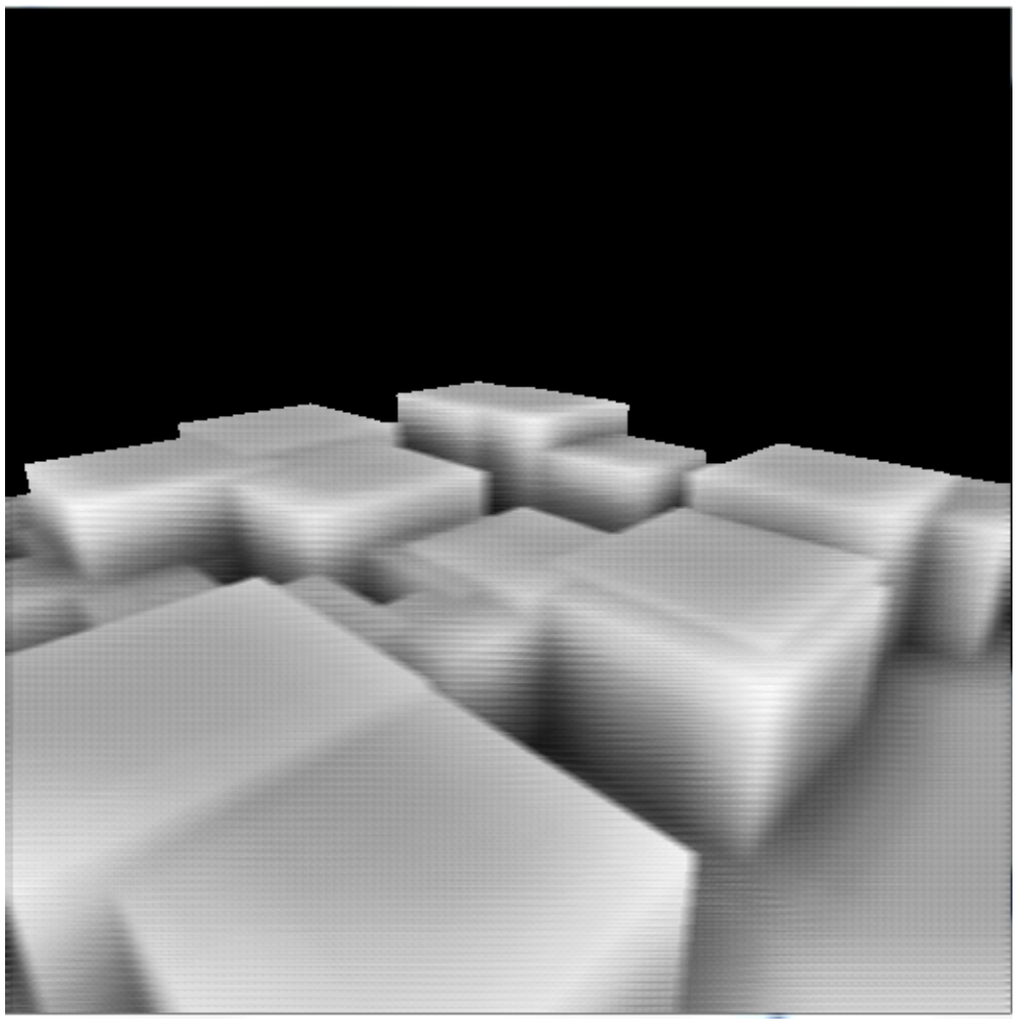




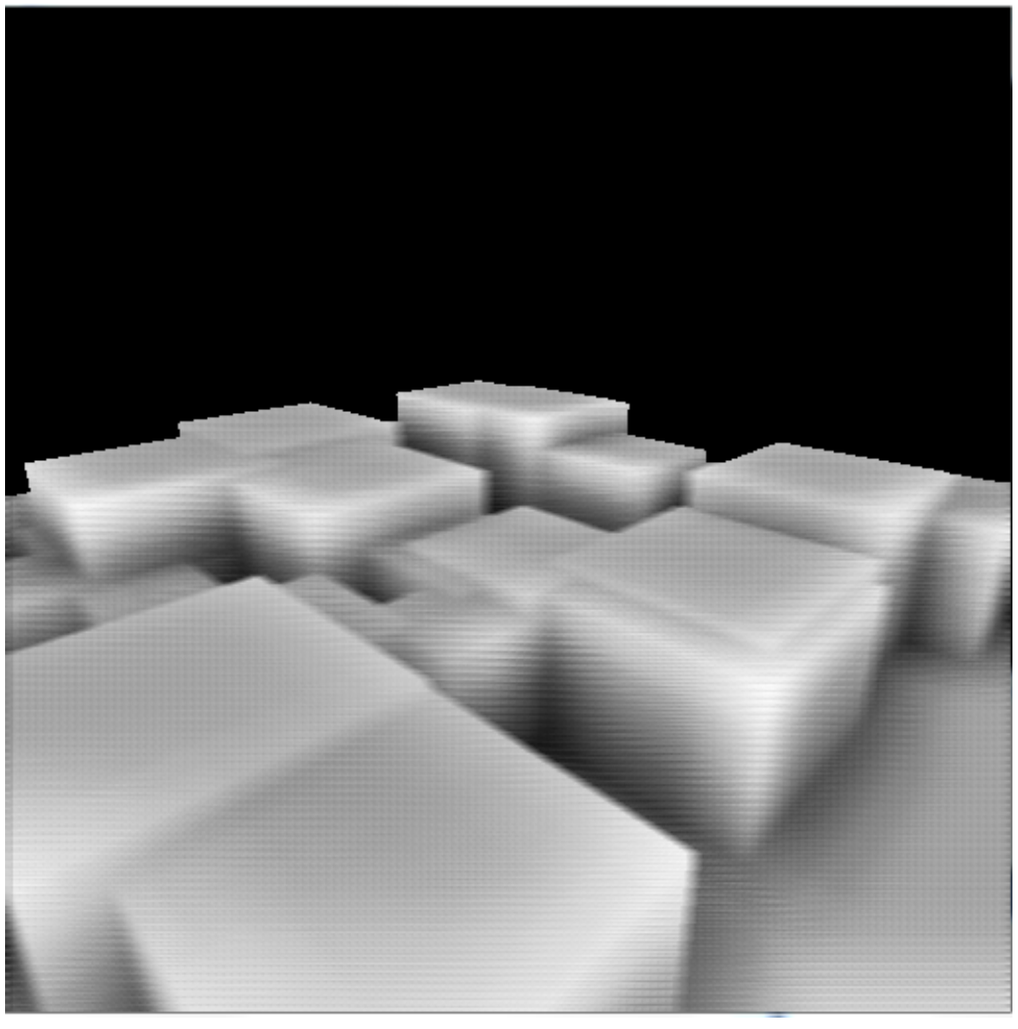
Banding artifacts

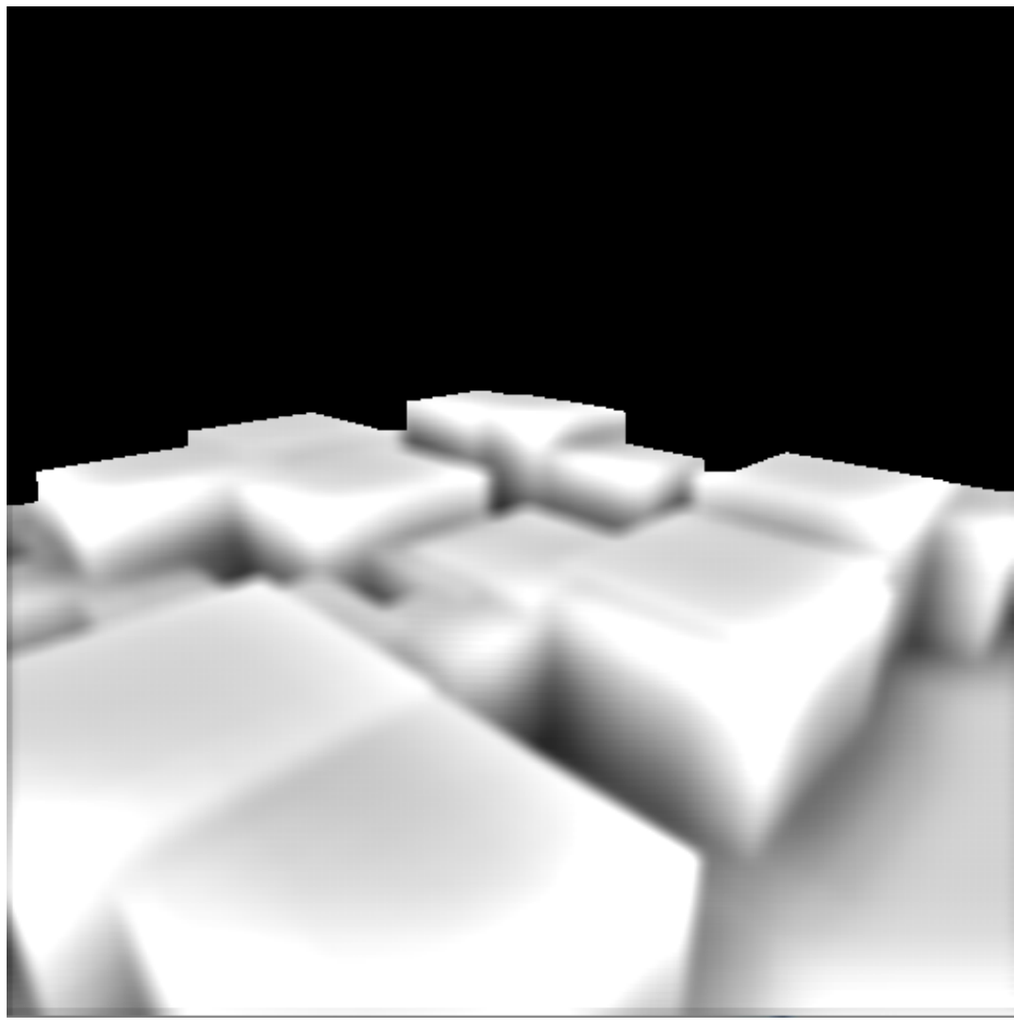
Blurring the Blocking Factor

Blurring the Blocking Factor

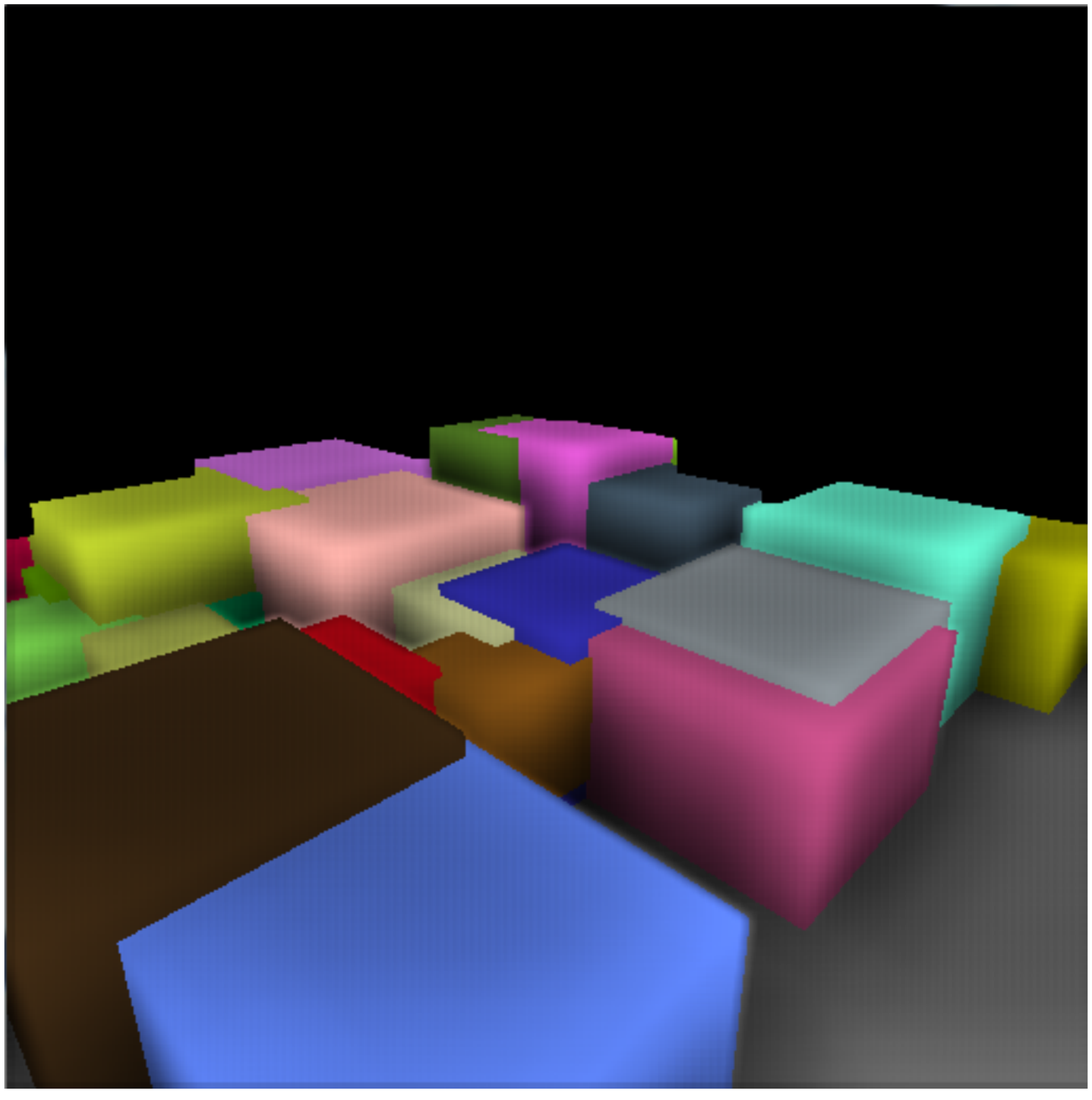


Blurring the Blocking Factor





Combining with the Color Values



SSAO Examples

http://www.cyberhead.de/?dev

Crytek - CryEngine 2

Crytek - More Examples

http://video.google.com/videoplay?docid=-2592720445119800709#

Crytek - More Examples

http://video.google.com/videoplay?docid=-2592720445119800709#

Crytek - More Examples

http://video.google.com/videoplay?docid=-2592720445119800709#

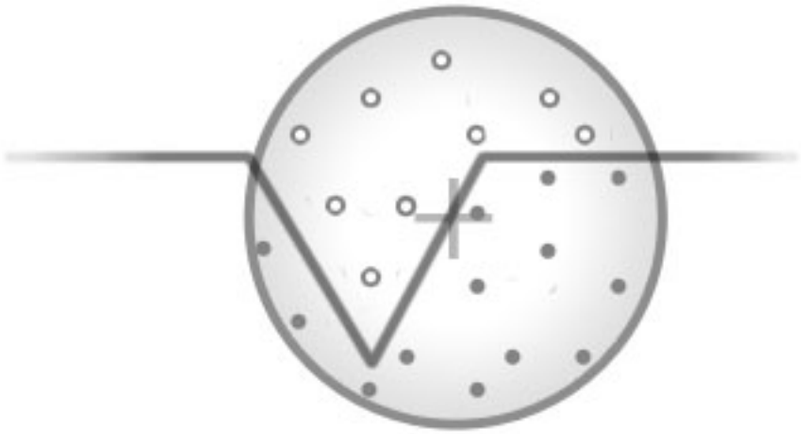
“Up” - Pixar

http://www.awn.com/articles/3d/academy-honors-renderman-getting-point

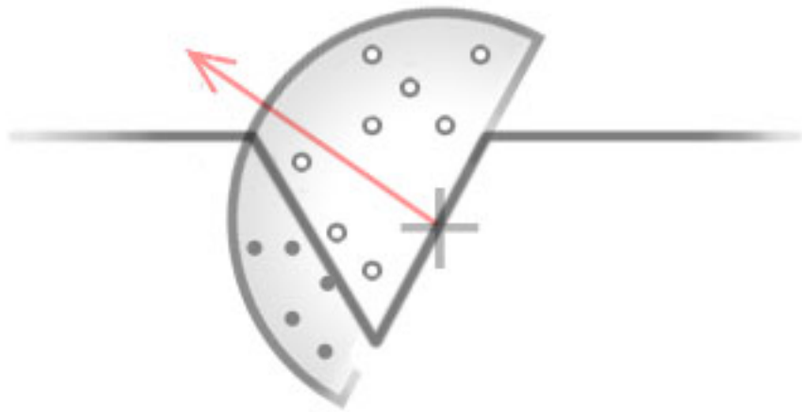
nVidia SDK Demo

Download the Demos from developer.nvidia.com

Extensions



Crytek Method (Full Sphere)



Normal-oriented Method (hemisphere)

https://www.youtube.com/watch?v=-IFxjKT7MXA

References

- SSAO Tutorials / Introduction
  - https://learnopengl.com/#!Advanced-Lighting/SSAO
  - http://john-chapman-graphics.blogspot.com/2013/01/ssao-tutorial.html
- Advanced topics (artifacts in SSAO)
  - https://mtnphil.wordpress.com/2013/06/26/know-your-ssao-artifacts/

Thank You.