Fast Real-time Caustics from Height Fields

Cem Yuksel and John Keyser
Texas A&M University

Motivation

Motivation

Motivation

- Caustics are important


no caustics

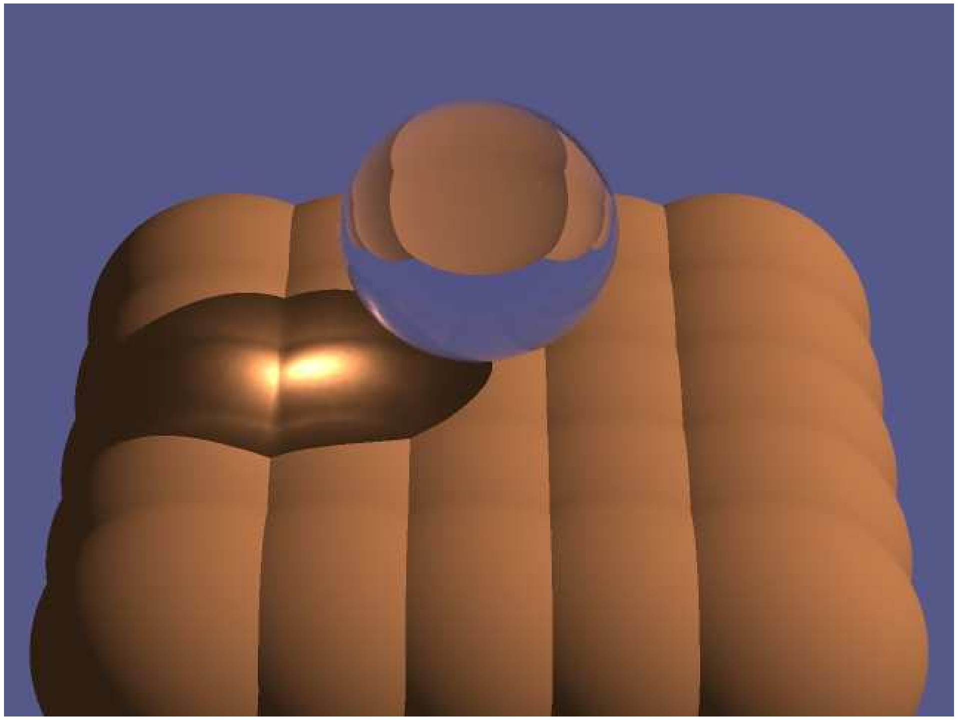

with caustics

Motivation

- Caustics are important
- Caustics are SLOW!
- Current real-time systems
  - Fake caustics
  - No caustics
- Real-time caustics
  - Only in tech demos
- We need a FAST technique!

Previous Work

- Monte Carlo path tracing
  [Kajiya 1986]
- Wavefront propagation
  [Mitchell and Hanrahan 1992]
- Backward ray tracing
  [Arvo 1986]
- Photon mapping
  [Jensen 1996]



Photon mapping – Image courtesy of Henrik Wann Jensen

Previous Work

- Caustics maps

  [Szirmay-Kalos et al. 2005]

  [Wyman and Davis 2006]

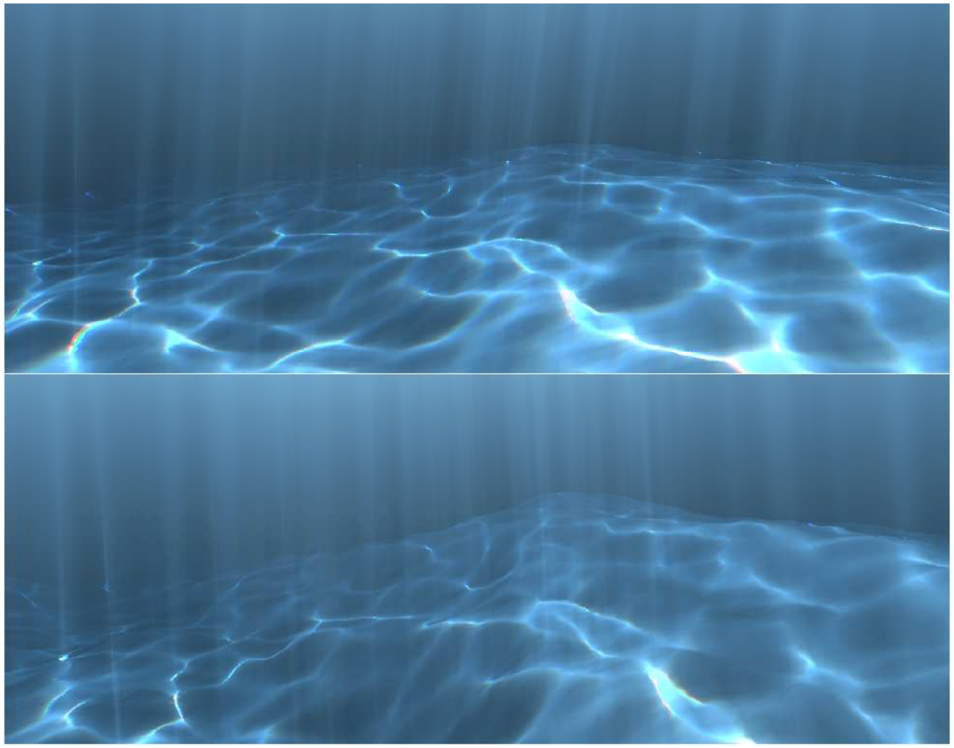
  [Shah et al. 2007]

  [Wyman 2008]

Hierarchical caustic maps – Image courtesy of Chris Wyman

Previous Work

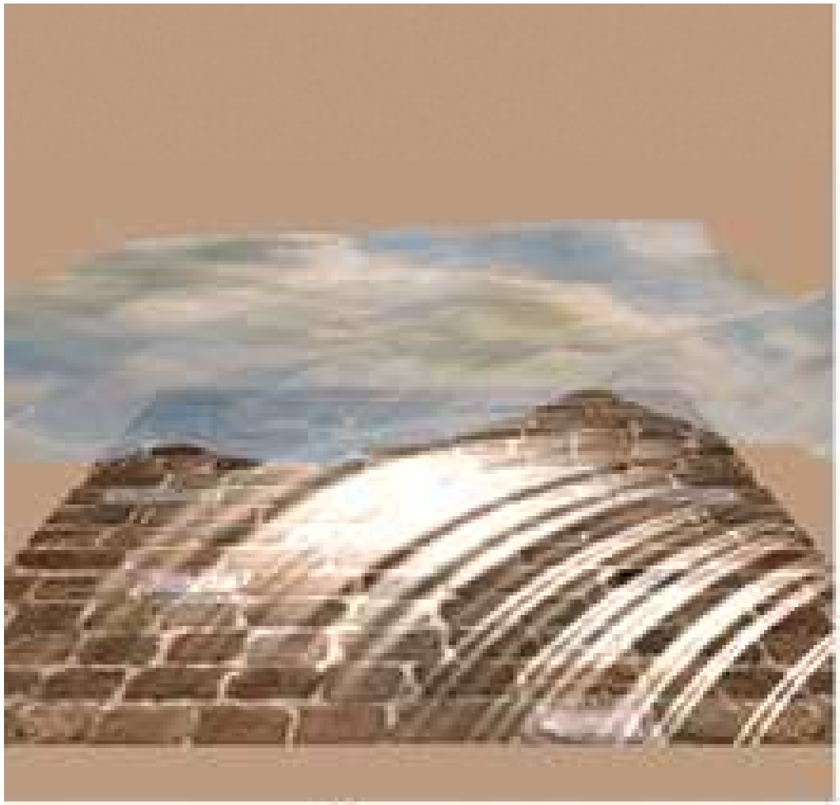
- Caustic textures
  - [Stam 1996]
- Beam Tracing
  - [Heckbert and Hanrahan 1984]
  - [Watt 1990]
  - [Nishita and Nakamae 1994]
  - [Iwasaki et al. 2001]
  - [Ernst et al. 2005]



Interpolated Warped Volumes – Image courtesy of Ernst et al.

Previous Work

- Rendering Water Caustics – GPU Gems
  [Guardado and Sanchez-Crespo 2004]



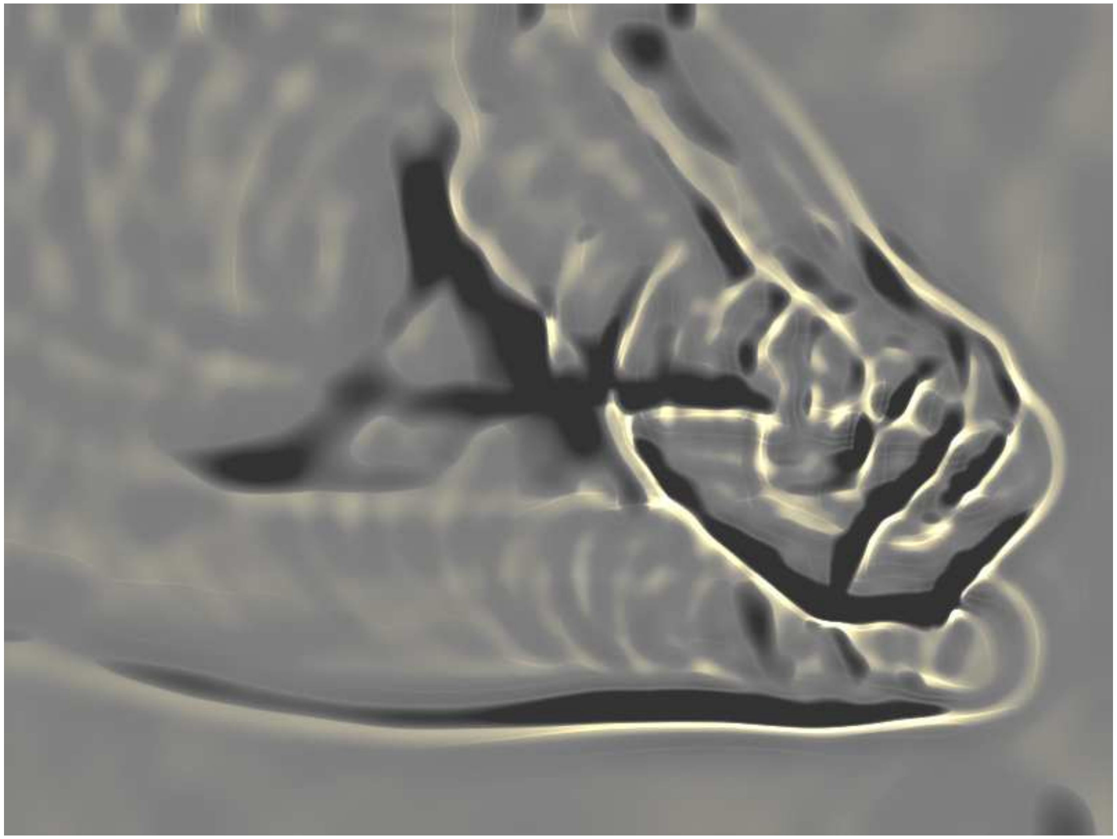
Hierarchical caustic maps – Image courtesy of Guardado and Sanchez-Crespo

Our Solution

- Fast real-time caustics
- From a height field surface
- Onto a planar surface

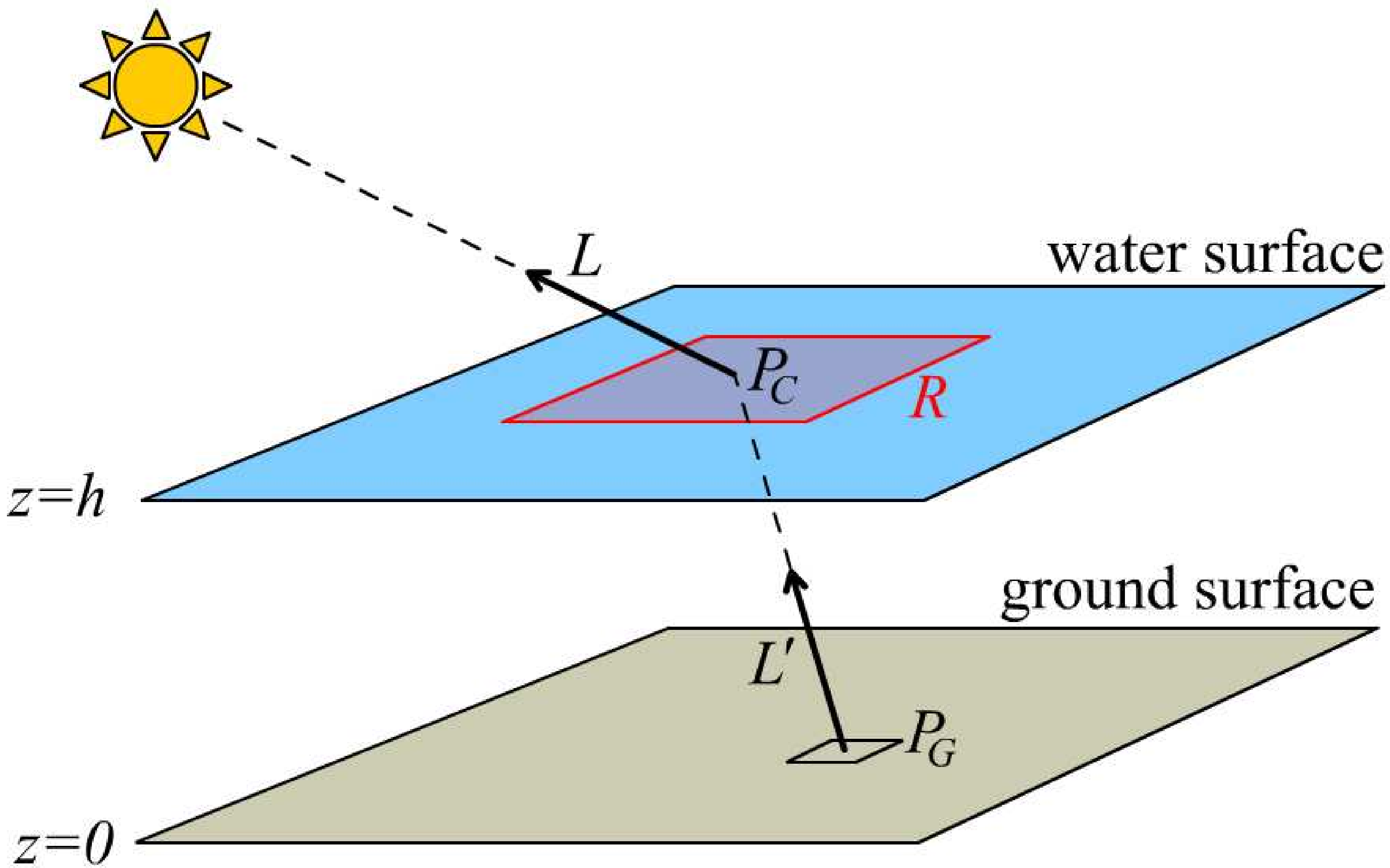
Caustics Computation

- Starting from the caustic-receiving surface
  - Flat plane
  - A caustic map that is mapped onto this plane



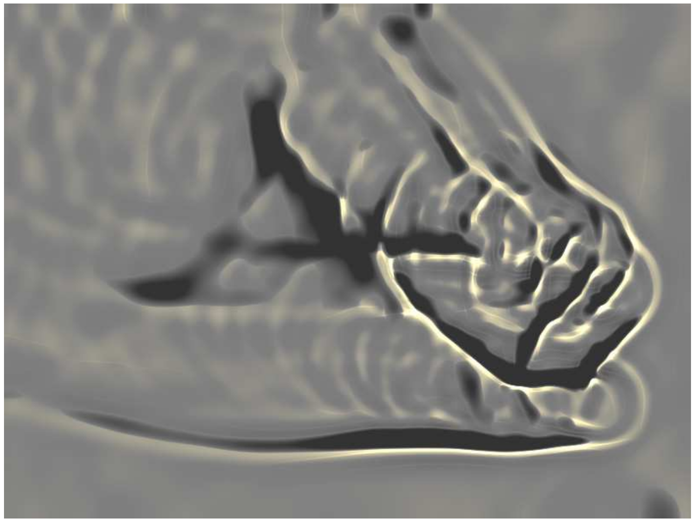
Caustics Computation

- For each pixel, sum refracted radiances toward the pixel



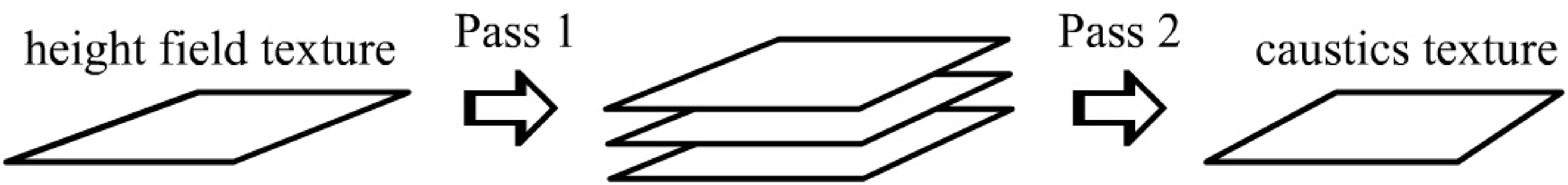
Caustics Computation

- Accurate as long as R is large enough
- Less accurate when the height field has
  - Large and
  - High frequency deformations
- Too small R → Underestimation
- Most simulations require very small R



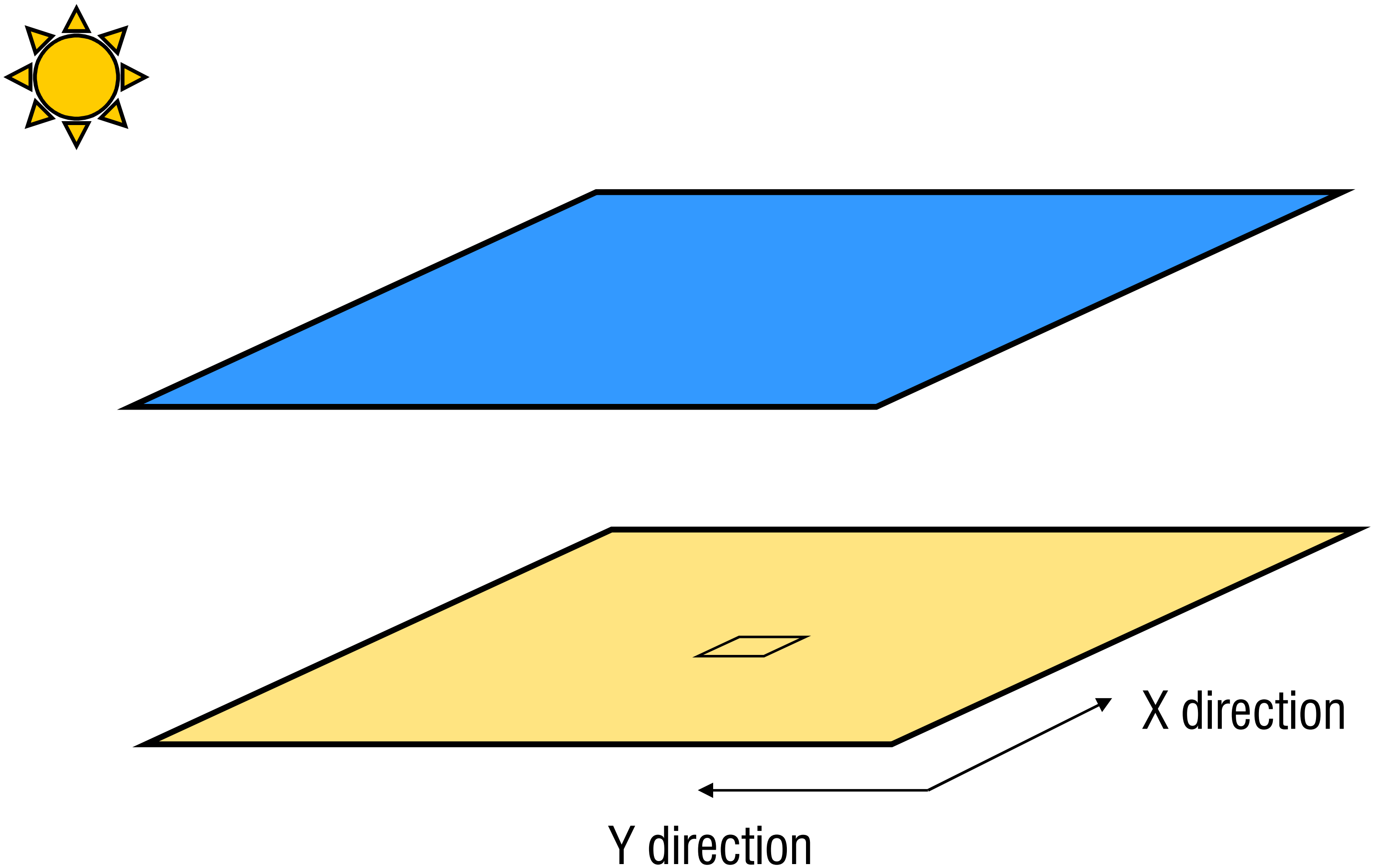
The Two-Pass Algorithm

- Similar to separable convolution filtering
- Pass 1: caustics in X direction
- Pass 2: caustics in Y direction



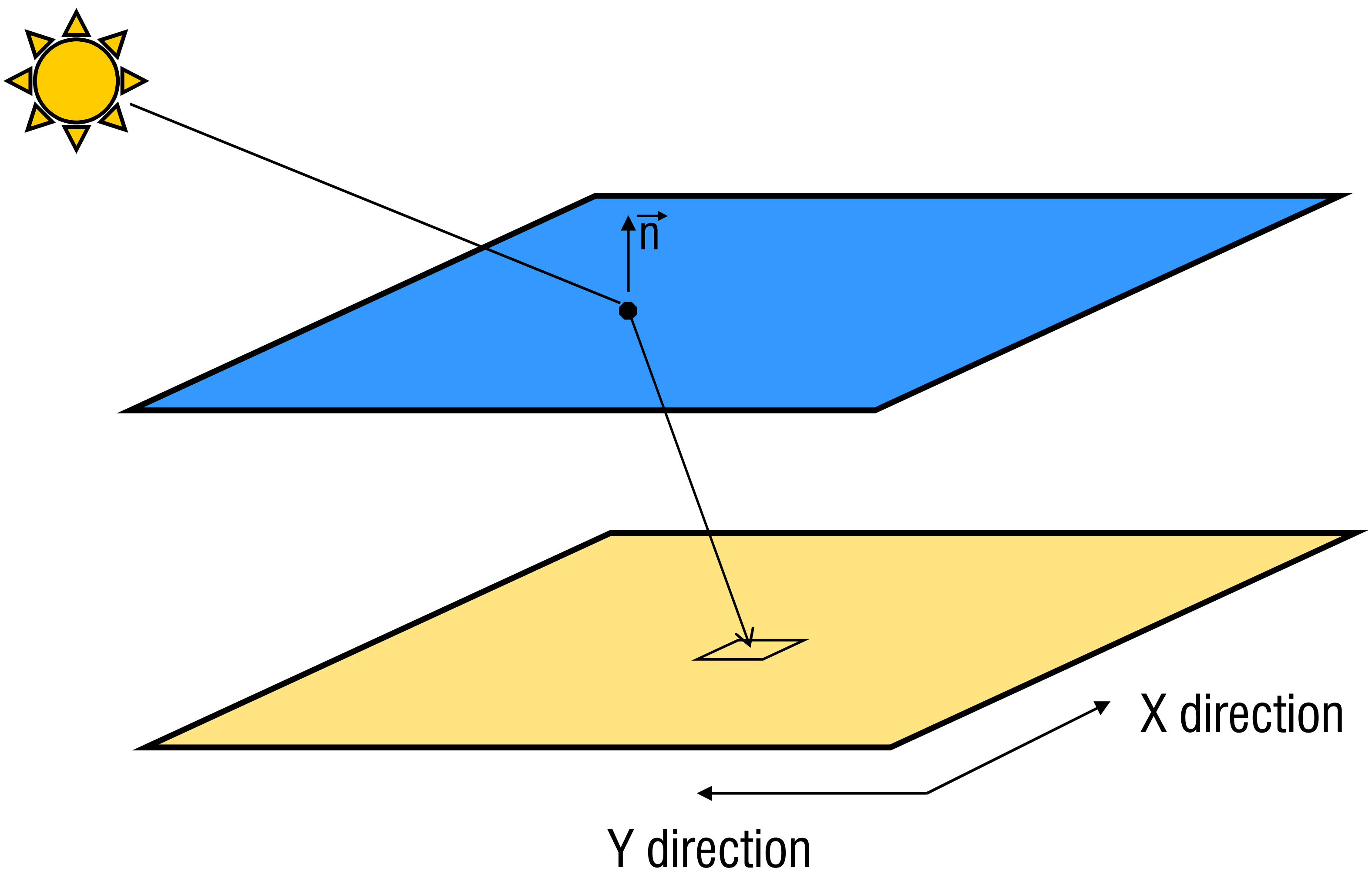
The Two-Pass Algorithm

- Pass 1:



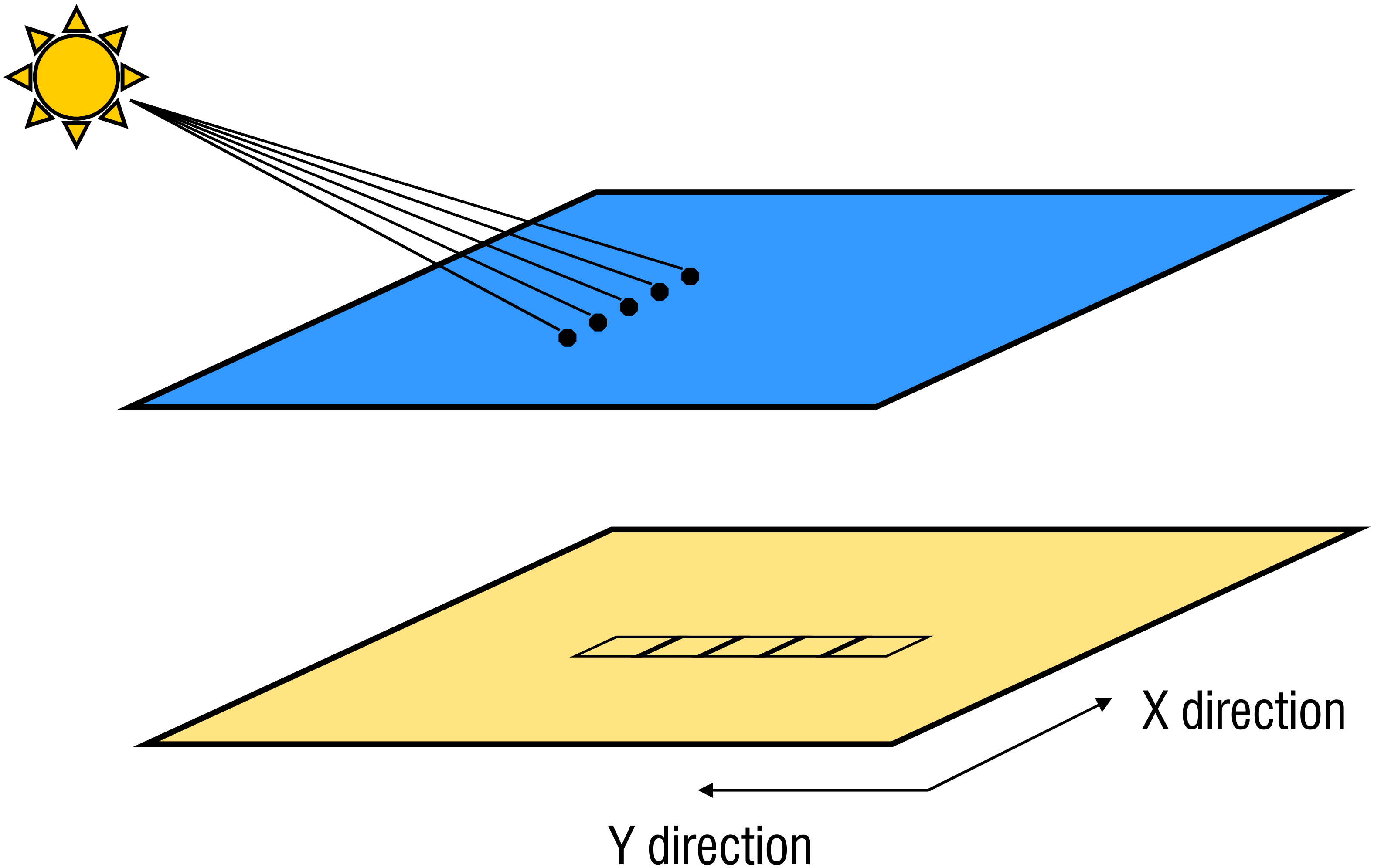
The Two-Pass Algorithm

- Pass 1:



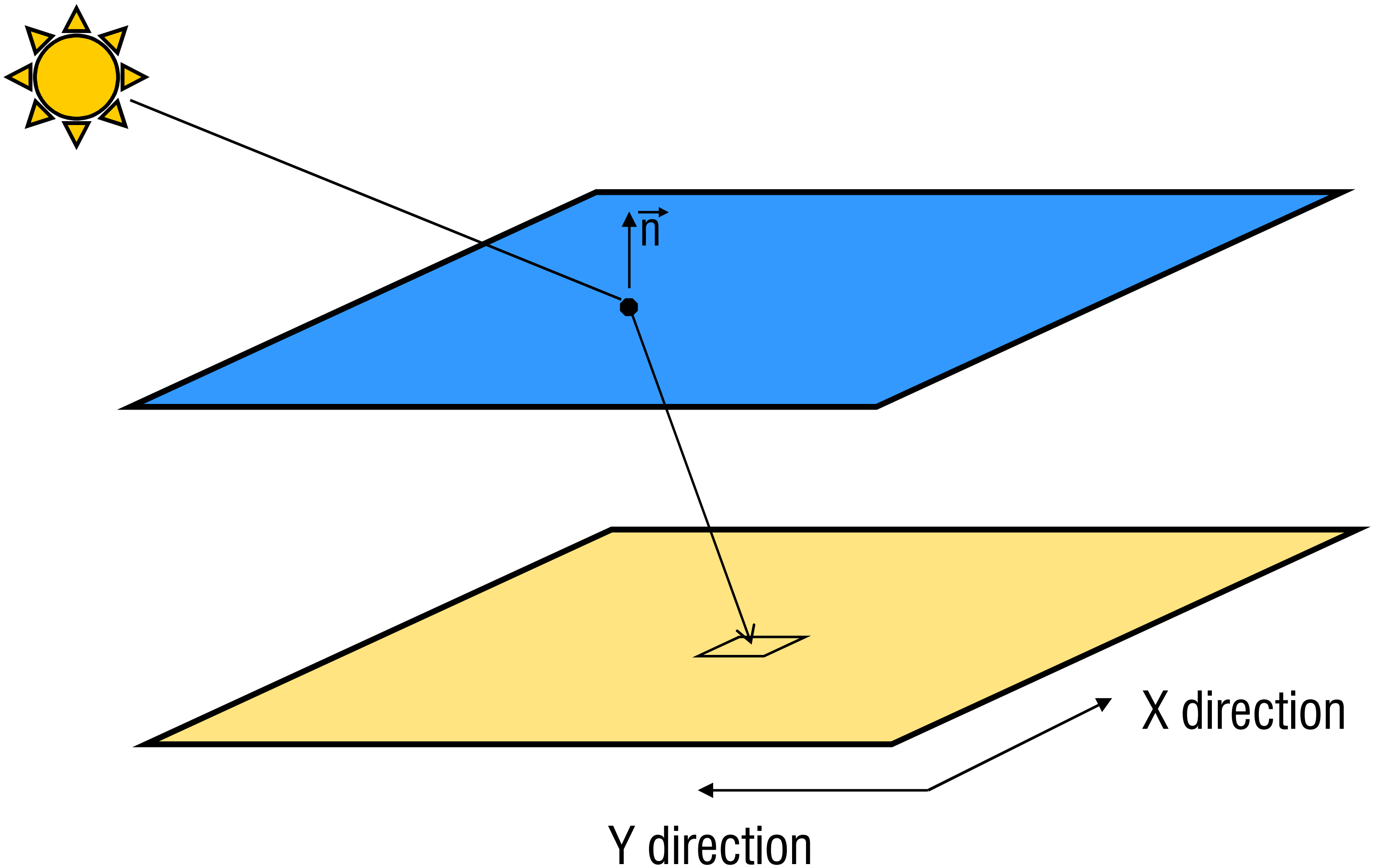
The Two-Pass Algorithm

- Pass 1:



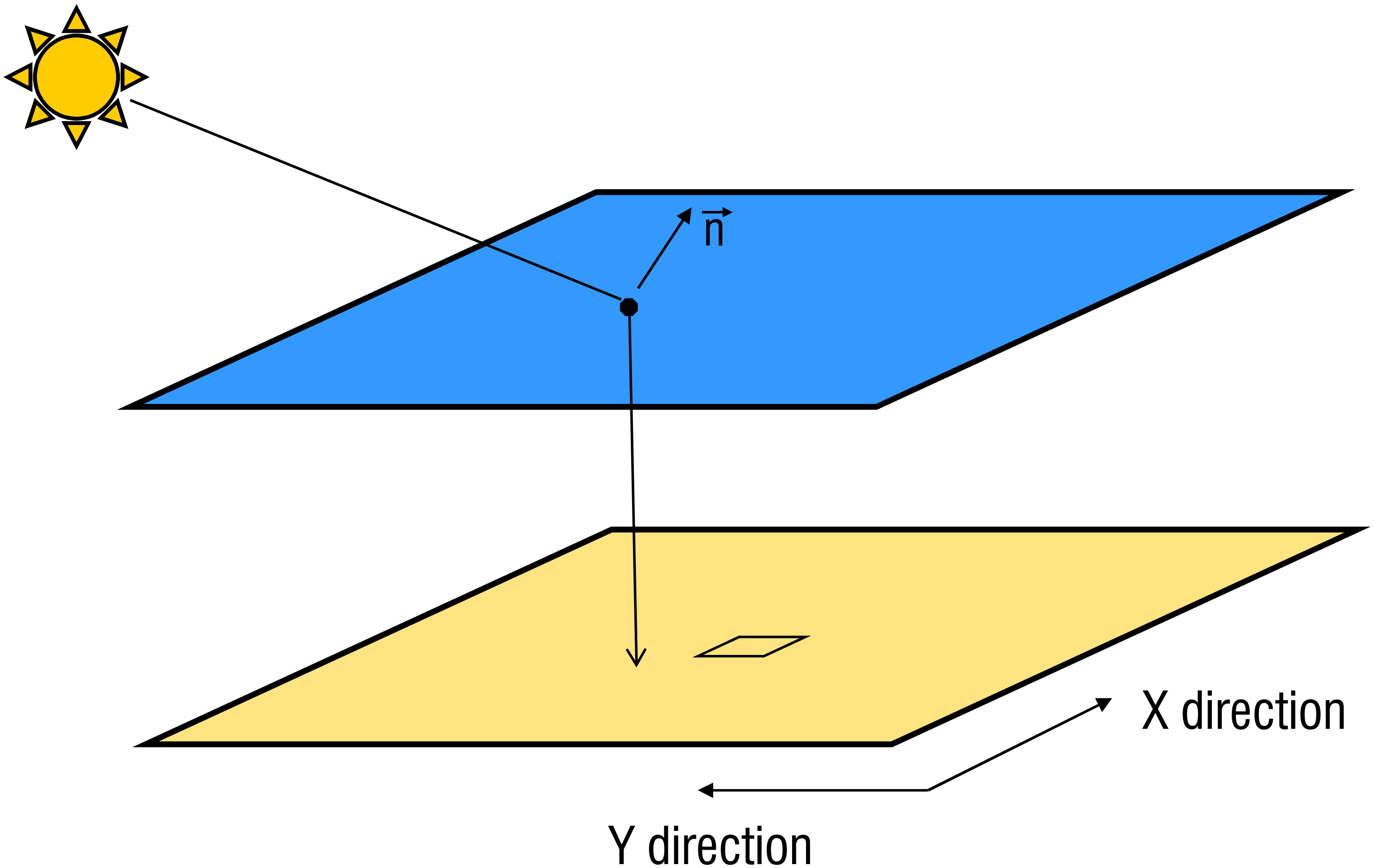
The Two-Pass Algorithm

- Pass 1:



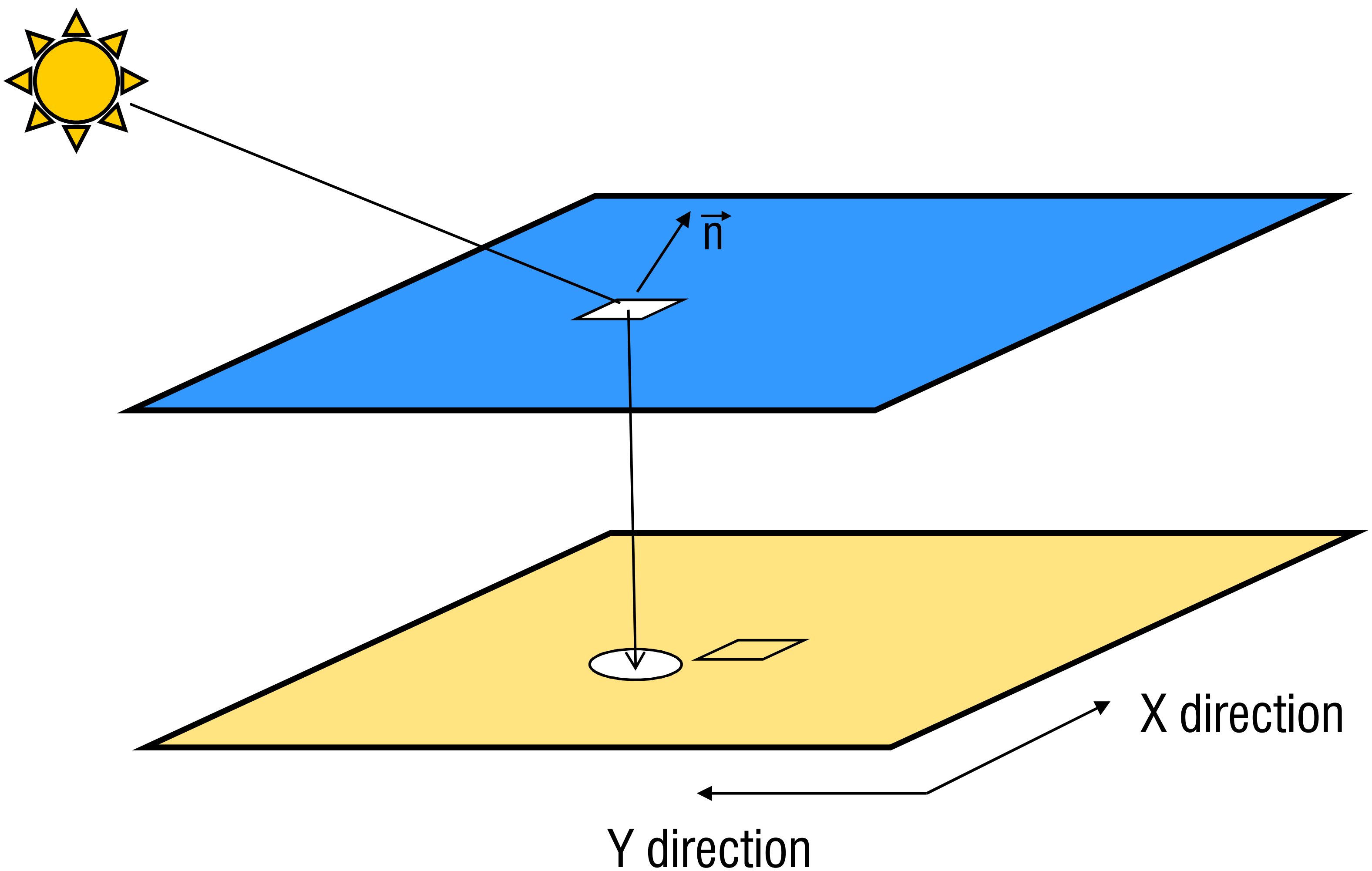
The Two-Pass Algorithm

- Pass 1:



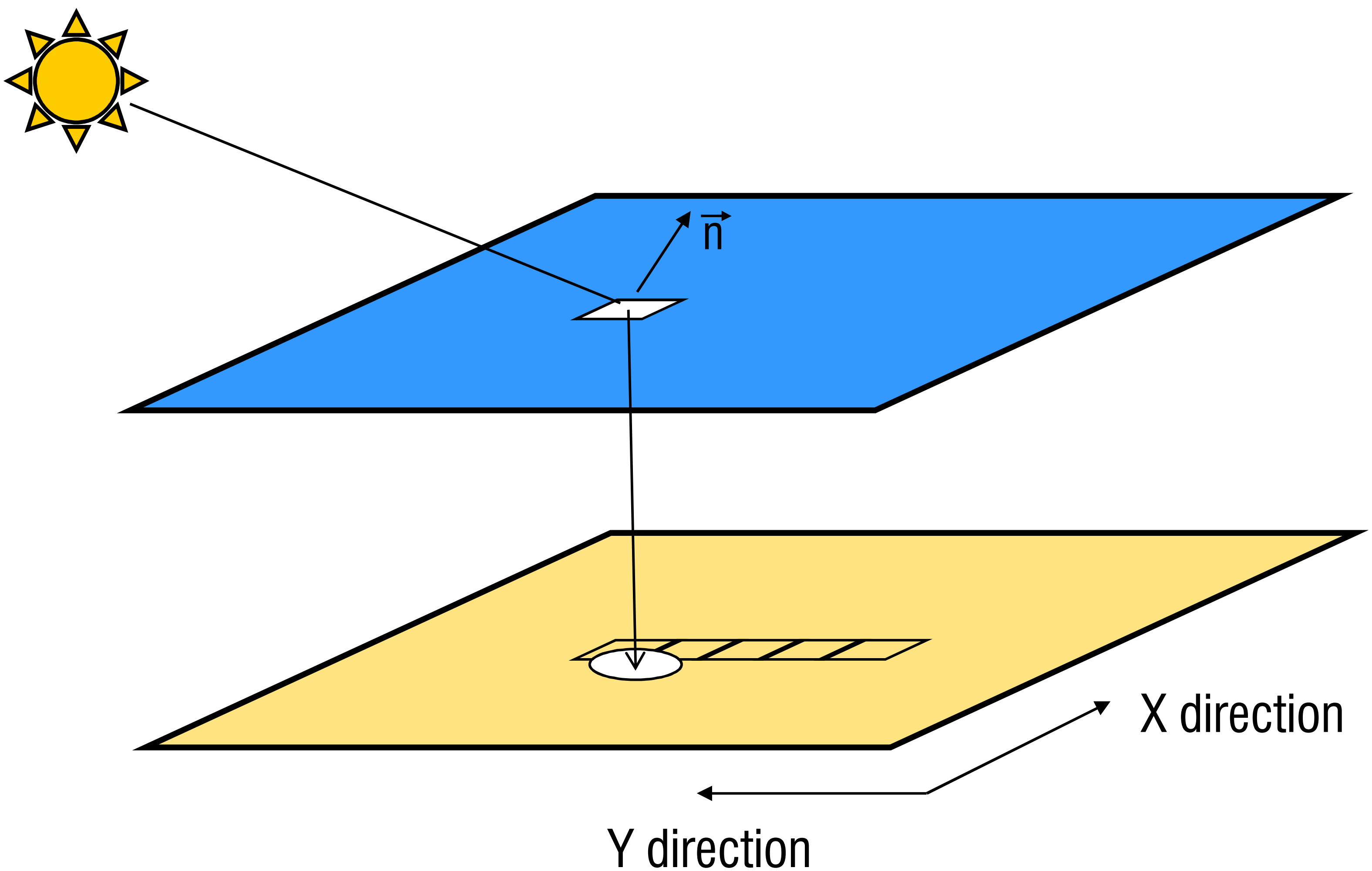
The Two-Pass Algorithm

- Pass 1:



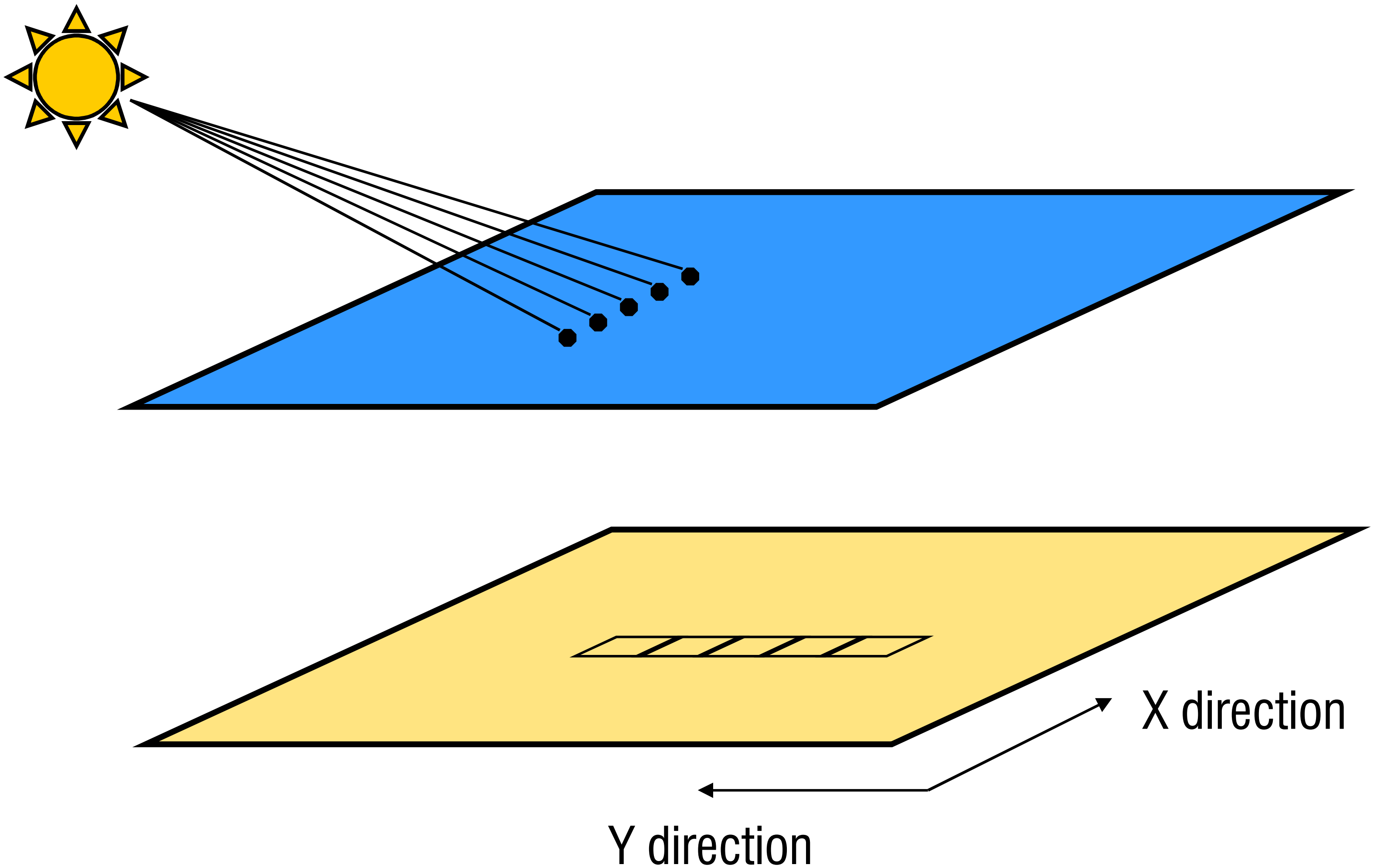
The Two-Pass Algorithm

- Pass 1:



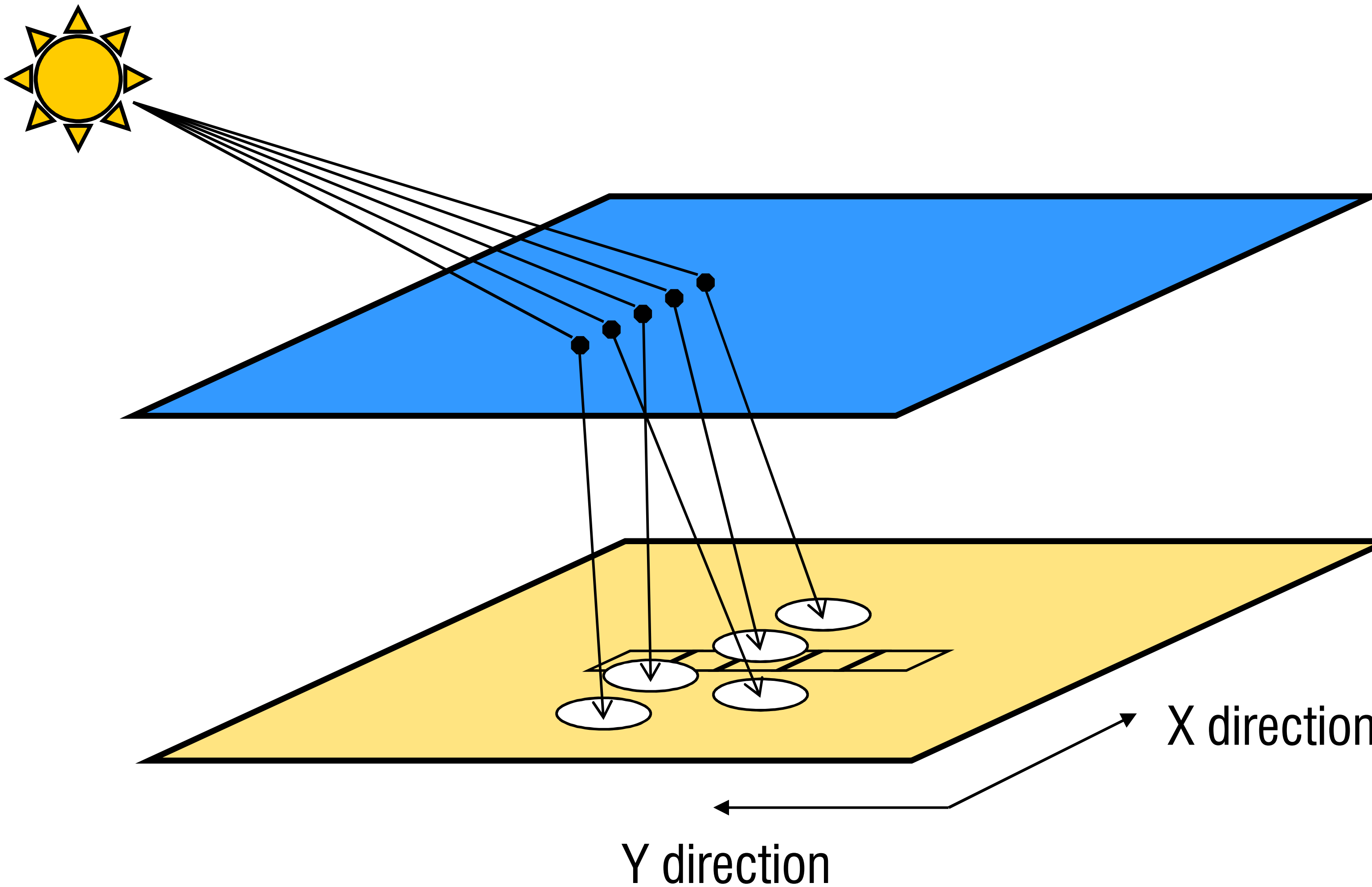
The Two-Pass Algorithm

- Pass 1:



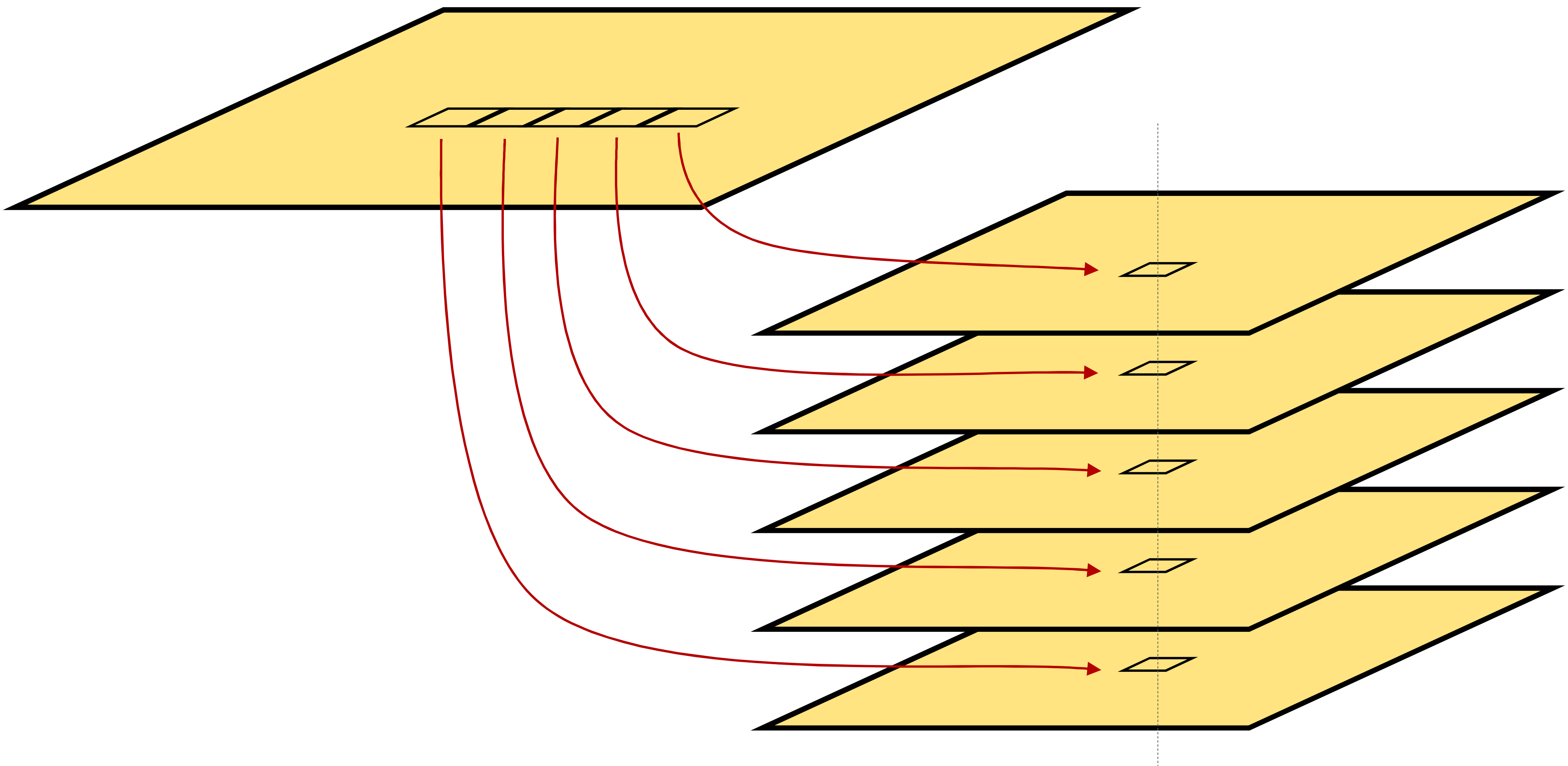
The Two-Pass Algorithm

- Pass 1:



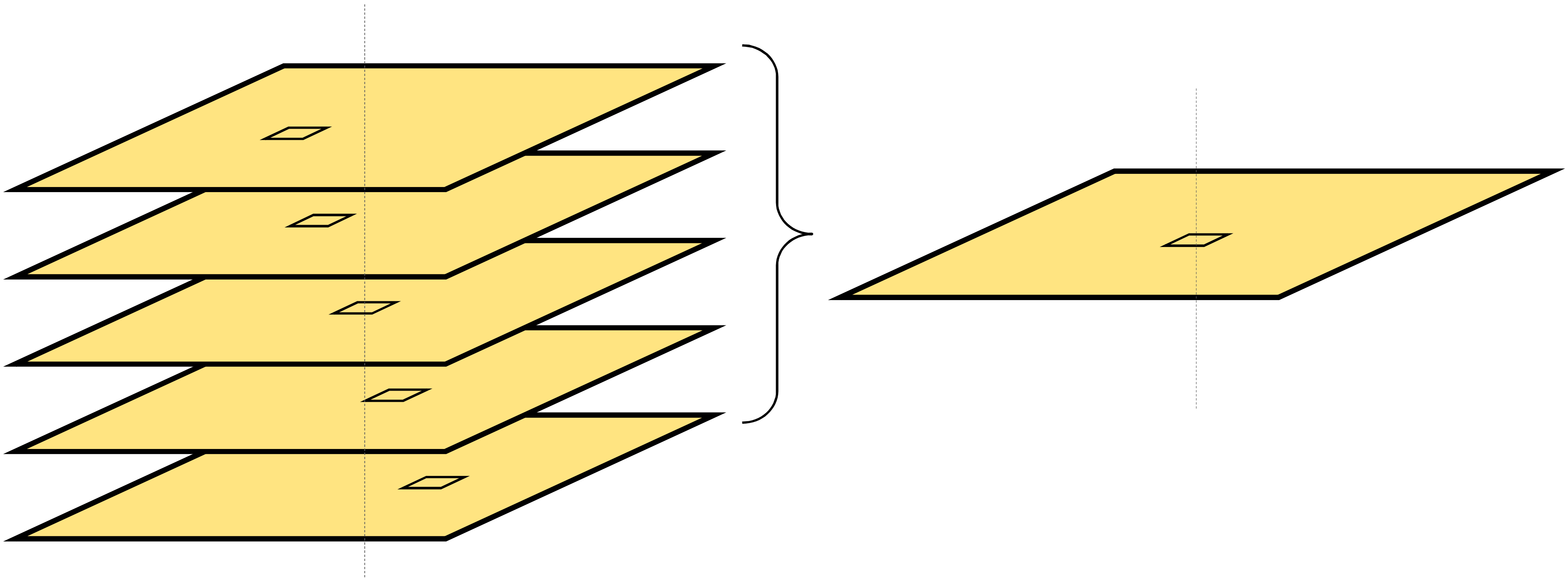
The Two-Pass Algorithm

- Pass 1:



The Two-Pass Algorithm

- Pass 2:
  - Accumulate caustic values at different textures



Implementation

- The whole computation is in Fragment Shaders
- Repeated computations in Pass 1
  - Refracted ray directions
- Speed-up
  - Introduce an additional pass before Pass 1
  - Precompute refracted ray directions
- Pseudo codes for Pass 1 and Pass 2 are in the paper.

Implementation

```
void Pass1(  out Pass1Out Out,
             in float2 P_G : TEXCOORD0,
             in float2 P_C : TEXCOORD1,
             uniform sampler2D heightField )
{
    // initialize output intensities
    float intensity[N];
    for ( int i=0; i<N; i++ ) intensity[N] = 0;
    // initialize caustic-receiving pixel positions
    float P_Gy[N];
    for ( int i=-N_HALF; i<=N_HALF; i++ ) P_Gy[i] = P_G.y + i;
    // for each sample on the height field
    for ( int i=0; i<N; i++ ) {
            // find the intersection with the ground plane
            float3 pN = P_C + ( i - N_HALF ) * xDirection;
            float2 intersection = GetIntersection( heightField, pN );
            // ax is the overlapping distance along x-direction
            float ax = max(0, 1 - abs(P_G.x - intersection.x));
            // for each caustic-receiving pixel position
            for ( int j=0; j<N; j++ ) {
                        // ay is the overlapping distance along y-direction
                        float ay = max(0, 1 - abs(P_Gy[j] - intersection.y));
                        // increase the intensity by the overlapping area
                        intensity[j] += ax*ay;
            }
    }
    // copy the output intensities to the color channels
    Out.color0 = float4( intensity[0], intensity[1], intensity[2], intensity[3] );
    Out.color1 = float3( intensity[4], intensity[5], intensity[6] );
}
```

Implementation

```
void Pass2(  out float4 color : COLOR,
             in float2 P_G : TEXCOORD0,
             uniform sampler2D inColor0,
             uniform sampler2D inColor1 )
{
    float val = 0;
    val += tex2D( inColor0, P_G + float2( 0, -3 ) ).r;
    val += tex2D( inColor0, P_G + float2( 0, -2 ) ).g;
    val += tex2D( inColor0, P_G + float2( 0, -1 ) ).b;
    val += tex2D( inColor0, P_G ).a;
    val += tex2D( inColor1, P_G + float2( 0, 1 ) ).r;
    val += tex2D( inColor1, P_G + float2( 0, 2 ) ).g;
    val += tex2D( inColor1, P_G + float2( 0, 3 ) ).b;
    color = val;
}
```

Results

Final Points

- Advantages
  - Works fast!
  - Does not require high-res water surface
  - Sequential texture access – cache friendly
- Limitations
  - Water surface must be a height field
  - Receiving surface must be a plane
  - Underestimation when R is too small
- Non-planar receivers?
  - Can be approximated as planar
  - Better than no caustics or “fake” caustics

Questions?

Fast Real-time Caustics from Height Fields

Cem Yuksel and John Keyser

Texas A&M University

no caustics

with caustics